2002

A new conceptual modeling framework for a distributed object-based industrial simulation system (DOBIS).

Bo. Gao
University of Windsor

A New Conceptual Modeling Framework For A Distributed Object-Based Industrial Simulation System (DOBIS)

By

Bo Gao. B. Eng.

A Thesis
submitted to the
Faculty of Graduate Studies and Research
through Industrial & Manufacturing Systems Engineering
in partial fulfillment of the requirements for the
Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada
2001

962051

I hereby declare that I am the sole author of this thesis. I authorize the University of Windsor to lend this thesis to other institutions, or individuals, for the purpose of scholarly research.

Bo Gao

I further authorize the University of Windsor to reproduce this thesis by photocopying, or by other means, in total or in part, at the request of other institutions, or individuals, for the purpose of scholarly research.

Bo Gao

The University of Windsor requires the signatures of all persons using or photocopying this thesis. Please sign below, and give an address and a date.

ABSTRACT

Different simulation applications require different levels of model detail. As the complexities and flexibilities of manufacturing systems are ever increasing, the flexibility of simulation system to model real world systems with arbitrary levels of detail becomes important. Existing simulation systems provide predefined models that limit their flexibility.

In this thesis, the author develops a conceptual modeling strategy that combines the Event-Scheduling strategy and the Object-Oriented paradigm. A basic element is developed to model any resource and is mapped to an extended Parallel DEVS formalism. This strategy provides general framework and system requirements for further computerized development. Such a system is able to implement recursive hierarchical modeling, which provides the environment to model real world systems at any level of detail. The basic element also provides simplified functions and syntaxes that make the system easy to learn and maintain. A two-layer object construction framework is also discussed to pushing down "low-level" aspects of simulation to the systems level, and away from the user. Several examples are developed in this thesis, which illustrate DOBIS has the potential to outperform existing systems in several aspects.

DEDICATION

To my parents,

To my wife

To my friends,

You are the wind beneath my wings

ACKNOWLEDGEMENTS

I would like to take this opportunity to deeply thank my supervisor, Dr. Filippo A. Salustri for his inspiration, encouragement, support and great guidance. I would further like to extend my thanks to the committee members, Dr. Kent, and Dr. Lashkari for sharing their helpful suggestions with me. I would also like to express my gratitude to Ms. Jacquie Mummery for her help. Finally, I would like to thank everyone that assisted and inspired me throughout, and made this accomplishment complete.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURES

Antimessage: Two messages that are identical in all fields but of opposite signs (+ and -) are antimessages to each other [26].

Assemble: Two or more parts are put together to form one entity.

Class: A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind [36].

Data Encapsulation: Data encapsulation describes the hiding of data structures and the implementation of procedures called methods to operate on the data of an object [2].

Disassemble: One entity is broken down into two or more subassemblies or parts.

Discrete Event Simulation: It concerns the modeling of a system as it evolves over time by representing the changes as separate events. This is the opposite of Continuous Simulation where the system evolves as a continuous function (differential).

Entity: In the context of this research, an entity is anything that is to be processed by the simulated system. It interacts with the resources. In industrial and manufacturing domain, it can be workpiece, part, assembly, etc. It is also referred to as flow entity.

Event-Scheduling, Activity-Scanning and Process-Interaction Strategy: These are all modeling strategies that define how a simulation system views the real world. Hence, the modeling information and modeling ability they provide are different. The generally accepted definitions of these strategies are:

Event-Scheduling Strategy: User defines the actions that every object will take

once an event is executed. The system's time advances by the timestamp of the current event.

Activity-Scanning Strategy: Similar to the above strategy but with conditional actions. The system has to continuously check whether any of these conditions and executes the actions if they have been satisfied. The system's time is advanced by fixed time interval.

Process-Interaction Strategy: User defines the life history of the flow entity, the sequence of its interactions with the resources in the system. The system's time is advanced by the timestamp of the current event.

Fidelity: A measure of how closely the simulation approximates the real world [35].

Inheritance: Inheritance is a technique for deriving new classes from existing ones through creating a subclass. A subclass inherits both data and methods of an existing superclass (sometimes called a parent). Inheritance permits a designer to abstract common features to a higher-level class [2].

LVT: Local Virtual Time. Used as a component in the Time Warp algorithm.

Method: A method is a function (subroutine) associated with an object [36].

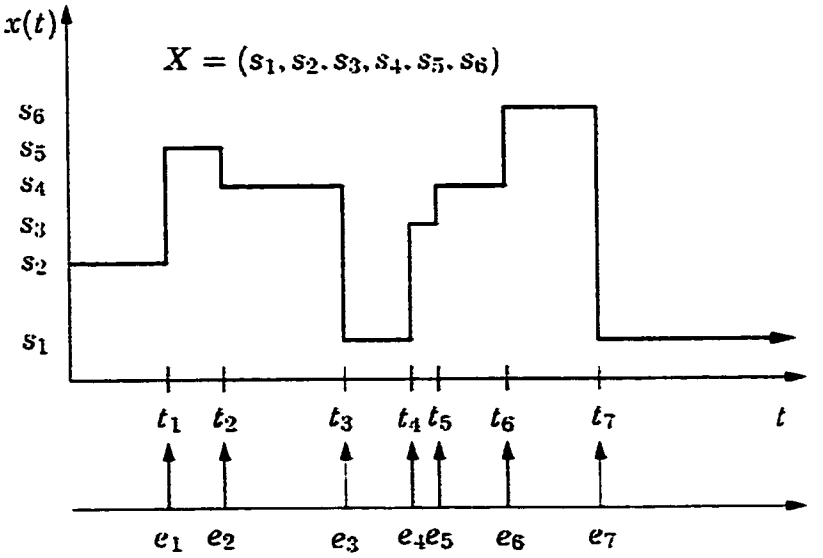
Model: A model is a simplified representation of a system (or process or theory), which intends to enhance our ability to understand and to predict, and possibly control the behaviour of the system [34].

Object: An object is an abstraction of something (tangible or intangible) that encapsulates information about itself, as well as interactions it undergoes within the system [37].

Object-Oriented: The term *object-oriented* is generally used to describe a system that deals primarily with different types of objects, and where the actions the system can take depend on what type of object the system is manipulating [43].

OO Simulation System: Object-Oriented Simulation System. It is the simulation system that is constructed by Object-Oriented programming language and benefits from the Object-Oriented concept in modeling.

Piecewise Constant: In discrete event system, state trajectory is a piecewise constant function that jumps from one value to anther when an event occurs. See the following figure: the trajectory of x(t) is piecewise constant.



Polymorphism: Polymorphism is the ability to take more than one form. Through polymorphism, the same method results in different behavior depending on the object to which it is bound [2].

Pseudocode: An outline of a program, written in a form that can easily be converted into real programming statements. Pseudocode cannot be compiled nor executed, and there are

no real formatting or syntax rules. The benefit of pseudocode is that it enables the programmer to concentrate on the algorithms without worrying about all the syntactic details of a particular programming language [43].

Resolution: Resolution in simulation is a question of detail. It is the level of detail that the simulation can input and output [35].

Reuse: Reuse is associated with the ability of using the same software elements for several purposes in different applications [2].

PE: Proto-Element. This is the basic form developed by the author to model any objects and functions in the industrial and manufacturing systems domain. Please see Chapter 3 section 3.2.2 for detail explanation.

Process: In manufacturing, it is a continuous operation or treatment. [41]

Simulation: It is the imitation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system being represented [49].

Test Head: A mechanism that scans all the conditional cases to see if any one of them is satisfied. When one of the cases is satisfied, the simulation engine executes the main body of that case.

Thread: In programming, a thread is a part of a program that can execute independently of other parts. Operating systems that support multithreading enable programmers to design programs whose threaded parts can execute concurrently [43].

Traditional Simulation System: It is the simulation system that does not have Object-Oriented technique in it.

Validation: It is substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model. [47, 48]

Verification: It is to ensure that the computer program of the computerized model and its implementation are correct. [47, 48]

CHAPTER ONE

INTRODUCTION

Today's advanced manufacturing systems tend to achieve higher levels of flexibility, which makes the system more complex in terms of machine interactions [1, 16, 39, 42, 52]. Simulation provides solutions including decision-making support, system design analysis, performance analysis, bottleneck detection, etc., yet modeling of these systems has been a challenging task. One of the concerns is whether the simulation model is "correct", which is addressed by the model validation and verification [44 - 48]. Another concern is whether the real world systems are modeled with sufficient detail, in other words, whether the resolution of the model is appropriate for the application [65]. Although the goal of modeling is to reflect enough, but not too much, of the complexity of the real system, the possible levels of detail are infinite especially in some critical fields of study. The need for the level of detail is arbitrary. The modeling flexibility of the simulation system needed to meet the arbitrary requirement is hence important.

Davis and Hillestad listed applications for high and low-resolution models in [50]. Usually a low-resolution model is used to support management, which views the system as a whole; a high-resolution model is often used when analysis is focused on certain processes. Therefore, a model for productivity analysis will be very different from that for testing the utilization of a certain tool in a machining centre. The resolution concern has some relationships with model validation [47]. Hamilton, etc. [35] pointed out that only

valid high-resolution models would result in higher fidelity, and that users often want to substitute resolution for validation because it is easier for users to check if small parts of the simulation were working correctly. Al-Aomar and Cook discussed the importance of high-resolution model in machine control design in [65]. Therefore, for both application and validations purposes, the author believes that it is necessary to develop a simulation system that provides capability to model industrial and manufacturing systems with flexible resolutions.

Figure 1.1 illustrates the author's work as it relates to the overall industrial simulation process. Banks [12] defines the process of model building is to capture a series of mathematical and logical relationships concerning the components and the structure of the real-world system.

High-resolution models have longer run time than low-resolution models [44]. The execution of a simulation will become an issue when the application must support real-time decision-making. Therefore, it is necessary to address this concern when developing the simulation system.

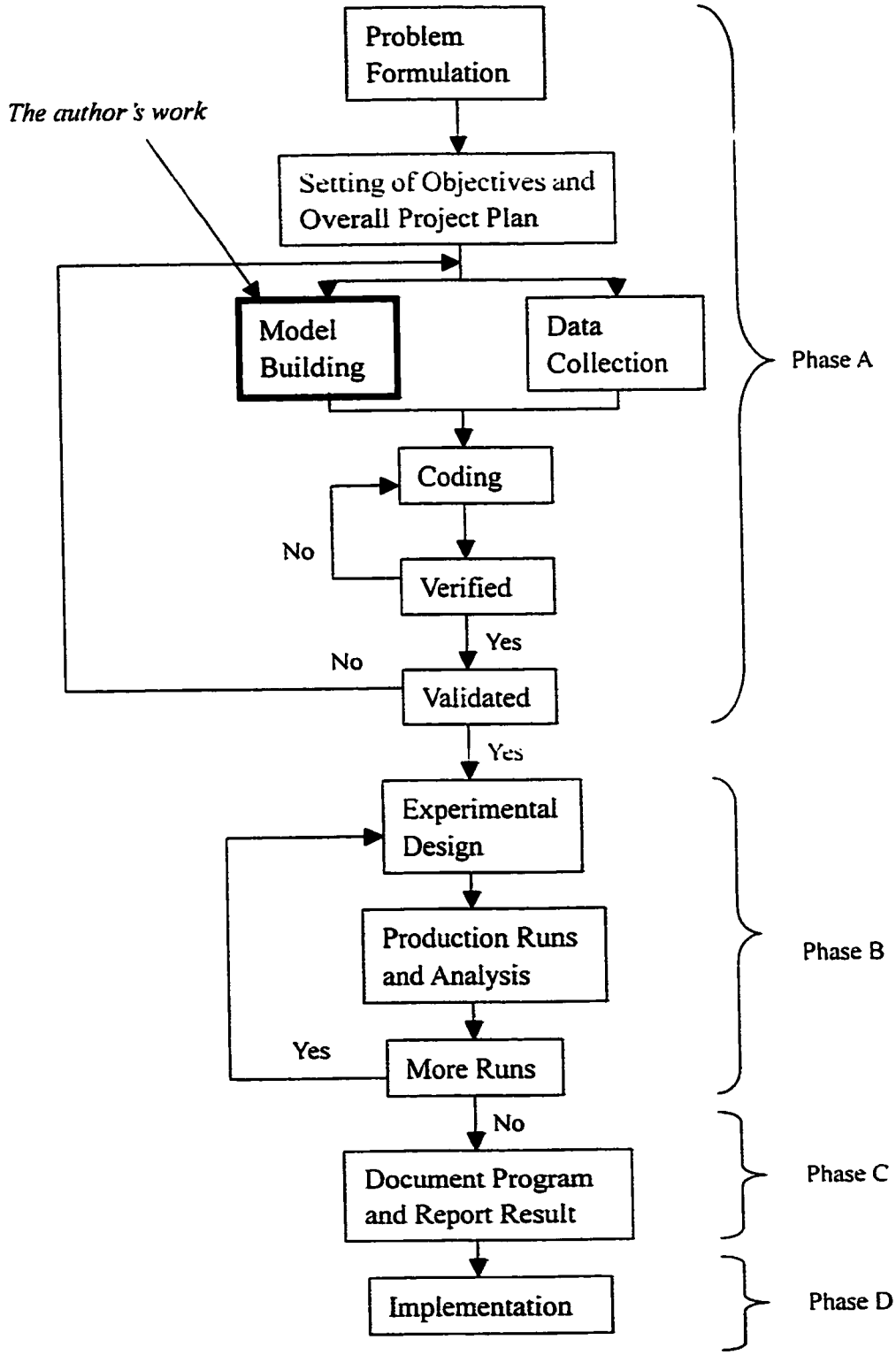


Figure 1.1. The author's work in relation with a simulation project (based on [49])

1.1 BACKGROUND

The author's work is a part of a larger project, the Intelligent Simulation System [82]. The goal of this project is to develop a new simulation system embedded with Artificial Intelligence. Simulations are often used to carry out "what-if" analyses, wherein a change in conditions is introduced into the simulation to see how the system would respond. Often these analyses result in changes to the system design, which then must be reflected in the simulation itself before the system can be tested. One can represent these steps as a loop:

1. Designing a system
2. Simulating it
    a) Model building (Phase A of Figure 1.1)
    b) Using the simulation to test system performance (Phase B of Figure 1.1)
3. Analyzing simulation results (Phase B of Figure 1.1)
4. Changing the system design to correct problems (Phase D of Figure 1.1)
5. Returning to step 2.

The idea of the project is that there are classes of problems for which steps 3 and 4 could theoretically be done by some sort of AI program, e.g., optimizing a factory floor to be able to handle certain kinds of failures. The goal, then, is to develop a software system

consisting of a simulation component and an AI component such that together they can automate the entire loop described above. In order to have maximum freedom to study this problem and develop the most appropriate AI technology, it is desirable to have a "home-grown" simulation package.

Therefore, the author's work is intended to lay the groundwork for that simulation package. In other words, it is to develop a new simulation system so that developers of this project will have more control over the code of the simulation system than they would have had if they had used an existing simulation program. The domain of interest is within the industrial and manufacturing systems domain, which is usually studied to improve throughput performances.

The author views the model of any manufacturing system as consisting of two levels of information, the system level and the resource level. At system level, routing information and interactions between resources are defined, whereas at resource level, internal operations and logical relationships between them, performances and resolutions of the model are defined. An internal operation is defined as the following by the author:

Definition: An Internal Operation is the operation that either directly or indirectly holds the workpiece or workpieces for a finite duration, and the length of the duration impacts the performance of the system being studied.

In the OO paradigm, since all the resources are treated as objects, the resource level

can be referred to as object level. Because the resources are working concurrently in nature, they are independent. Internal operations of a certain object are working on the same workpiece and therefore they are related by the workpiece, and therefore they have logical relationships. This is the fundamental difference between a resource and an internal operation.

Generally, objects are modeled in various degrees of detail, depending on their perceived importance with respect to the modeling objectives [64]. For example, a Flexible Manufacturing System (FMS) can be modeled in three levels of detail, as illustrated in Figure 1.2.

When modeling the FMS as a part of a whole production system, the FMS can be modeled as (a), in which the detail of the FMS is ignored. If the model (a) is treated as a system, model (b) can be developed to provide a relatively higher level of detail, in which the number of workstations, the functions of the workstations, the AGV and its routing information are modeled. If one of the workstations is treated as a system, its internal operations and their logical relationships should be captured as model (c).

Although existing simulation systems provide the ability to model real world systems to some level of detail, they have two main drawbacks: a) the maximum possible level of detail is limited to their predefined functions and model structures; b) the tailored model structures and functions that are "naturally mapped" from real-world objects employ complex syntaxes. It is not wise to have a huge system with predefined "high-end" resolution for every resource in the manufacturing system, because: a) the possible

required level of detail of the application is arbitrary; b) the required detail may not be covered by the predefined structure; c) the possible structures of resources are infinite; d) such a huge system is hard to learn and maintain, leading to high cost.

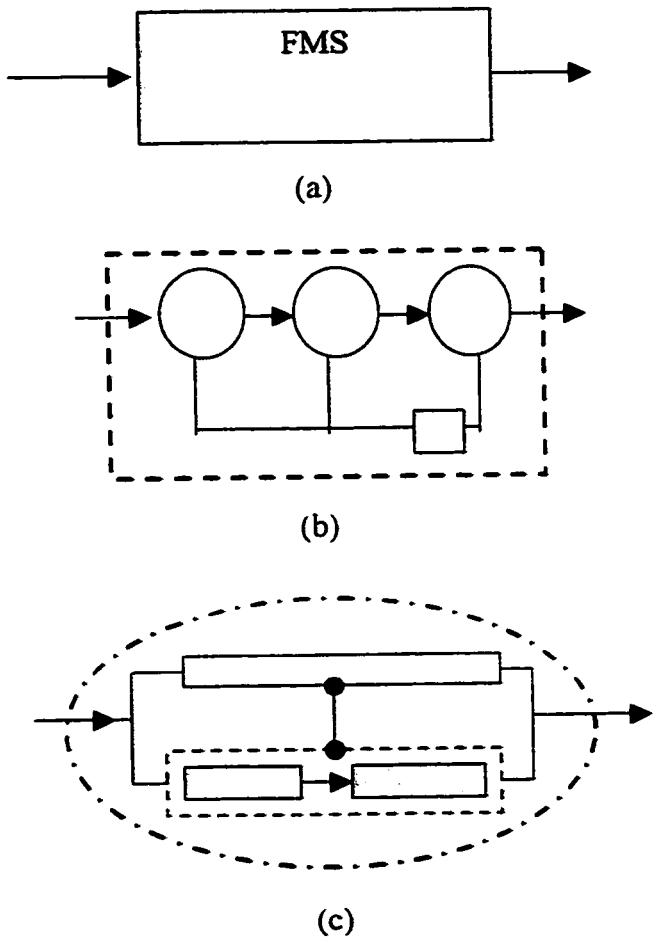


Figure 1.2. FMS modeled at different levels of detail

Rectangles are workstations, circle is automatic-guided vehicle, lines are guide-paths, arrows are directions of the flows of workpieces

The example illustrated in Figure 1.2 presents an interesting scenario: when increasing the resolution, the internal operations of model (a) emerge from object-level to

system-level in model (b). In other words, the object model (b) is the system model of itself. This relationship remains valid between (b) and (c). From this observation, it is shown that an object model can be detailed by treating it as a system model, leading the author to propose that the internal operation, object and system are different views of the same thing, therefore they can be modeled by a basic element that consists of both system-level and object-level information. and such basic elements can be structured to model complex resources with flexible resolution.

In order to prove this proposal, the author starts by studying the modeling strategies and methodologies, proposing a modeling strategy that combines the Event-Scheduling strategy and the Object-Oriented paradigm. Based on this modeling strategy, the author informally proposes a basic element that is capable of modeling all the resources and the internal operations with flexibility and simplified syntax. Then the author maps the proposed basic element into the rigorous Parallel DEVS formalism to provide formal representation. Finally. the author addresses the computing capability concern accompanied with the high-resolution models by employing the Time Warp distributed computing algorithm as an example for illustration.

1.2 STATEMENT OF THE THESIS

The author's thesis is that it is possible to develop a basic modelling element of uniform structure and simple syntax for industrial and manufacturing systems, that has

flexible resolution to improve the modelling capability of a simulation system.

1.3 ORGANIZATION OF THE THESIS

This thesis is organized as following: Chapter 2 reviews the current modeling strategies, methodologies and distributed computing algorithms. Chapter 3 presents the author's new framework. An example implementing the new framework is presented in Chapter 4. Conclusions are made in Chapter 5.

CHAPTER TWO

LITERATURE REVIEW

In this chapter, the author reviews the modeling strategies, methodologies and distributed computing in simulation.

2.1 THE STATE OF THE ART IN RESEARCH OF MANUFACTURING SYSTEMS SIMULATION

A comprehensive discussion of the state of the art in the research of object-oriented manufacturing systems simulation can be found in [2], as well as some insight about the underlying approaches applied by the researchers.

Often, it is confusing to say the modeling strategies and methodologies, and the Object-Oriented paradigm are all representing worldviews. Indeed, they all play key roles in providing a powerful and easy to use modeling environment. Usually a certain modeling methodology, e.g., Petri Nets, is proposed to answer questions within a domain [68], in other words, a modeling methodology that works for one type of questions dose not necessarily work for another. The Object-Oriented paradigm is usually applied on top of a modeling strategy to provide natural mappings, ease-of-use and other related advantages. The modeling strategies concern the viewpoint of the modeller, e.g., as flow entity or as resource. Therefore, the object-oriented paradigm is neither a modeling strategy nor a modeling methodology; it is an *abstraction methodology* instead. [2] did not discuss the

modeling strategies or methodologies. It discusses the advantages of applying object-oriented paradigm in constructing classes of objects that serve the modeling strategies.

To some extend, the modeling ability of a simulation system is determined by the abstraction ability of the simulation language [2]. Because the Object-Oriented (OO) paradigm's worldview, which is the foundation of the Object-Oriented Programming (OOP), provides many advantages in discrete-event modeling [56], applying OO in simulation modeling becomes one of the focuses in current research field. [2] concluded that OOP applied to manufacturing simulation modeling and analysis can be potentially very beneficial.

According to the current open researches, [2] gave a framework that is used to develop a simulation system. There are two fundamental phases: *development of architecture, implementation and application. Development of the architecture* results in a set of classes, methods, tools, etc., used to create and analyze specific simulation models. *Implementation and application* is the process of using the class hierarchies and methods to create specific simulation models in order to demonstrate and evaluate the classes and methods. The development of the architecture phase begins with a *domain*, employs a process of *analysis* and *modeling*, results in the *architecture*, a generic manufacturing system modeling formalism. The framework is shown in Figure 2.1.

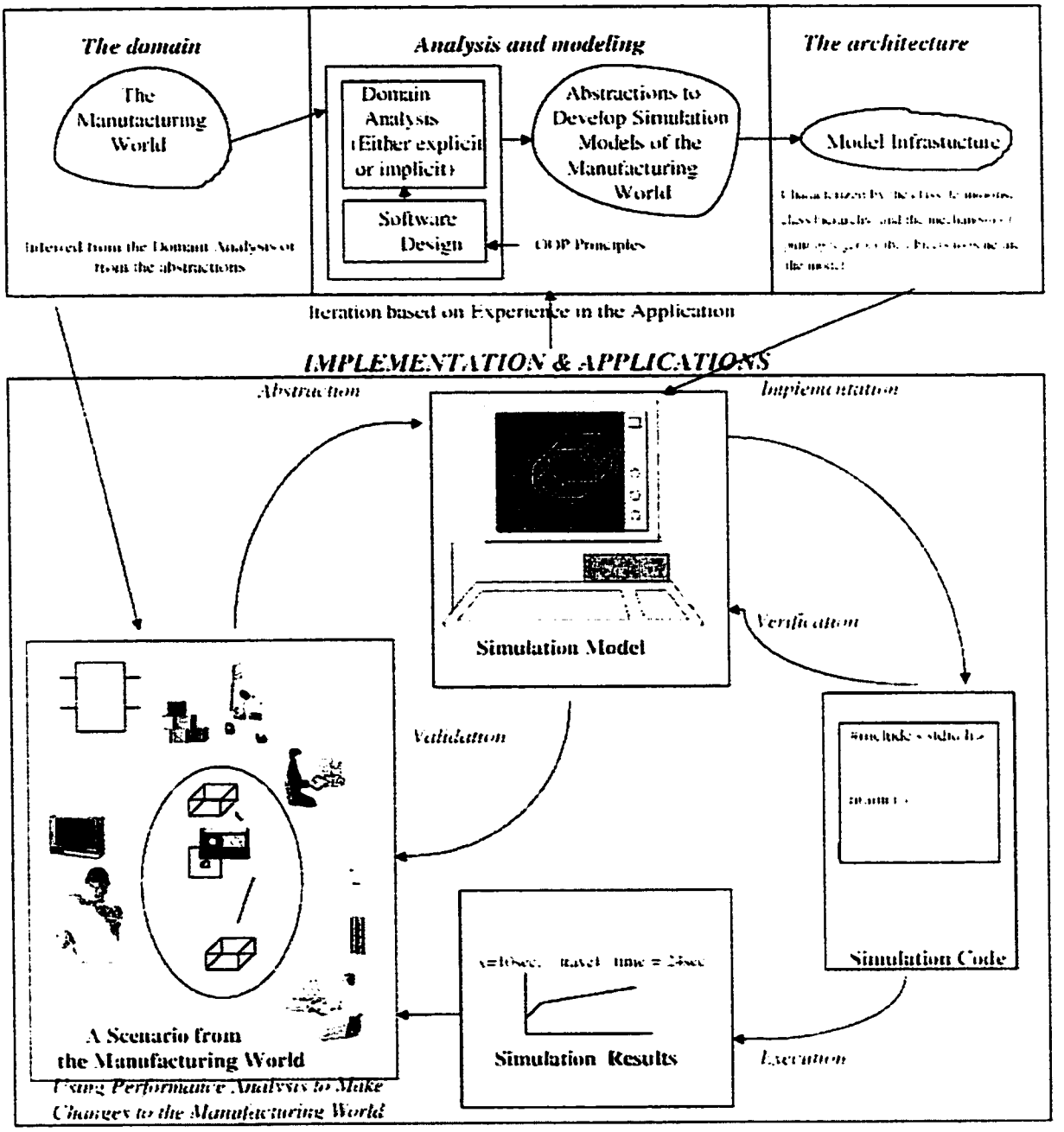


Figure 2.1. Development of the architecture of the simulation system [2].

Domain analysis, in this context, describes how researchers view or choose to talk about the manufacturing domain. The Architecture/Model Infrastructure is characterized by the class definitions, class hierarchies, and mechanism of putting together the objects to generate the model.

Table 2.1. Current major schools of systems, their research objectives and their OOP rationales [2].

System/Group	*Research Objectives*	*OOP Rationale*
BLOCS/M	Design a library of software modules to assemble special-purpose simulation models for manufacturing. Make the class library more reusable and easily comprehensible. Develop simulation models so they run efficiently.	Reusability. Ease of maintenance.
DEVS	Hierarchical and reusable model bases. Combining simulation modeling and AI techniques. Exploring distributed simulation models and architectures.	Exploring compatibility between OOP and discrete-event world-view. Reusability.
Laval	Develop an intelligent object-oriented model and simulation of manufacturing systems. Simplify the description of complex systems.	OOP provides a hierarchical world-view and polymorphism. Natural mapping.
OOSIM	Develop an object-oriented simulation modeling framework for representing the interactions between parts, automated processes, and operator problem solving for discrete manufacturing systems. Design a reusable library of classes to support modeling of manufacturing systems at different levels of abstraction from the viewpoint of material flow, control and supervisory control. Support real-time, interactive simulations.	Natural mapping Reusability.
OSU-CIM	Develop a modeling, analysis and optimization environment for manufacturing systems. Develop a modeling framework that permits the separate specification of physical, information and control elements. Develop formal methodologies for multi-level modeling and simulation model parallelizing. Design and implement an OOP-based modeling environment that permits programming-free model creation and multiple problem-solving approaches with a single base model.	Modularity and reusability OOP facilitates modeling at different levels of abstraction. Natural mapping
SmartSim/ SmarterSim	Produce a simulation environment that can be used by manufacturing engineers as a computer-aided design tool for the design of manufacturing systems.	OOP is useful in creating a simulation program generator. A good mapping is possible between system entities and icons in the software. Reusability.

Six major schools of systems are reviewed by [2], each of them has its own worldview and approach. This difference leads to different architectures. Under the different architectures, they have different objectives and rationales illustrated in Table 2.1. According to [2], the architectures are influenced, either explicitly or implicitly, by two distinct concerns: 1) the content and structure of the domain of intended application; and 2) software engineering. Together, these two concerns guide the decisions leading to specific

classes, class hierarchies, methods, and model implementation strategies.

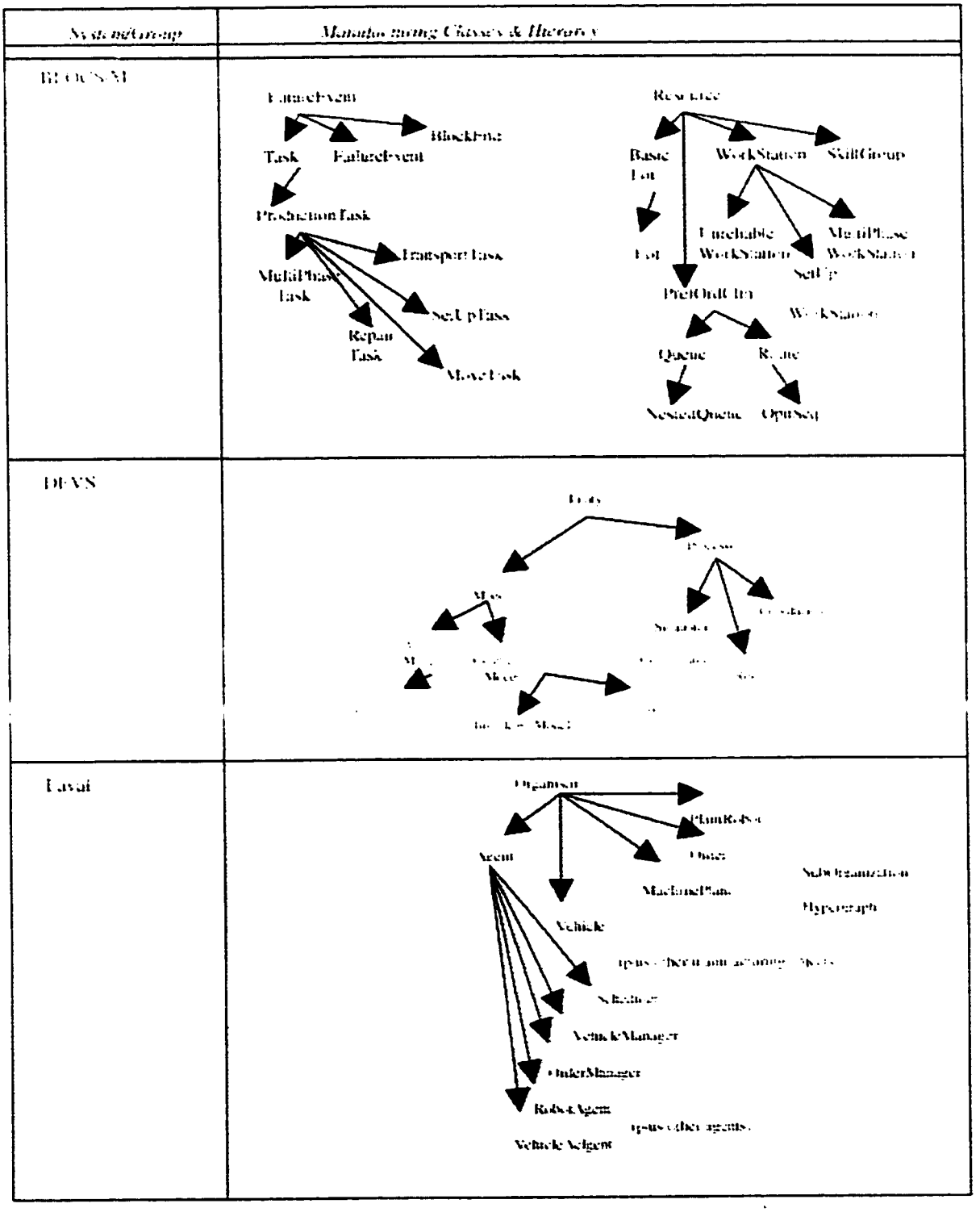


Figure 2.2. Manufacturing classes & hierarchy of existing Object-Oriented system [2]

(to be continued)

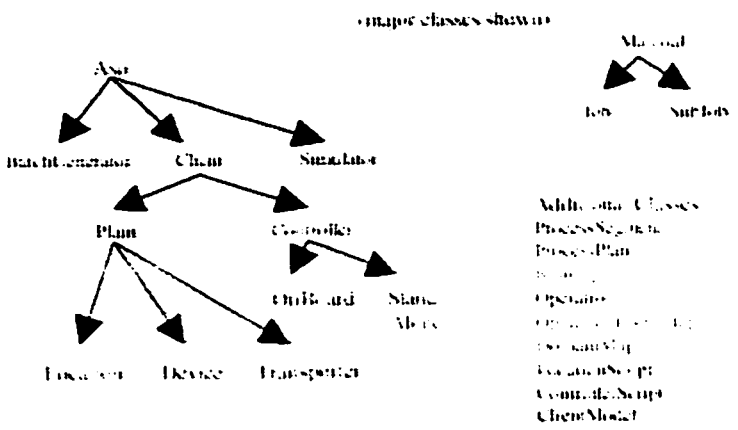
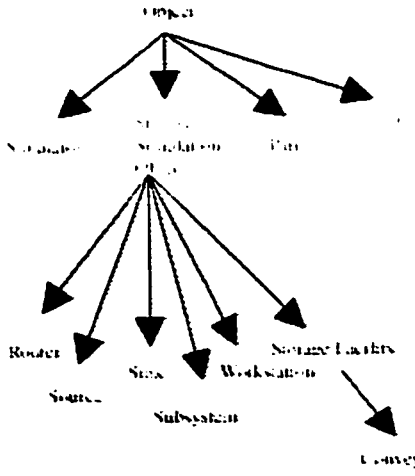
SystemGroup	*Manufacturing Classes & Hierarchy*
OOSIM	
OSU-CIM	Physical • Plant • WorkCenter • WorkStation • AssemblyWorkStation • MaterialHandler • Queue • CapacitatedQueue • Buffer Information • CustomerOrder • ShopOrder • Operation • Routing • BOM • ItemMaster Control • QueueController • AssemblyQueueController • WorkCenterController Note: The OSU-CIM architecture is composed of a collection of the classes shown. The hierarchy is typically does not compress many levels.
SmartSim/ SmarterSim	

Figure 2.2. Manufacturing classes & hierarchy of existing Object-Oriented system [2]

(continued)

Based on their distinct motivations, different class hierarchies are generated as illustrated in Figure 2.2.

BLOCS/M has four fundamental abstractions to represent objects in manufacturing: *lots*, *resources*, *tasks*, and *routes*. The fundamental abstractions in DEVS are *atomic models* and *components* built from the atomic models. The Laval effort views a manufacturing system as an organism composed of intelligent entities called *agents* and non-intelligent entities called *objects*. OOSIM classes are based on an explicit domain analysis of discrete-part manufacturing systems. This domain analysis results in the following fundamental abstractions: *material*, *locations*, *controllers*, and *process plans*. In OSU-CIM, entities in a manufacturing system have *physical* properties, *control* properties, and *information* properties. Like BLOCS/M, the OSU-CIM architecture emphasizes the software design principle of one function / one object to achieve reusability. SmartSim/SmarterSim researchers have applied the DEVS formalism to develop representations that correspond closely to manufacturing entities. The SmartSim/SmarterSim research builds on the fundamental abstraction of *subsystem* in DEVS to provide additional abstractions for *parts*, *workstations*, *conveyors*, and *routers* [2].

The develop environment dictated by the their platforms, and their applications are shown in Table 2.2.

Table 2.2. Platforms and applications of current researches [2]

[illegible]	Platform	Application domains
BLOCS/M	Objective-C and ICPak201, a software utilities library for interfaces.	Semiconductor manufacturing systems modeling; FMS for metal forming.
DEVS	SCOOPS, a superset of PC-Scheme on IBM PCs and TI Explorer.	Intelligent autonomous systems modeling (Autonomous robot simulation, printed circuit board test architecture)
Laval	Smalltalk-80.	Virtual cellular manufacturing system. Rolling mill application.
OOSIM	AT&T 2.1 C++ on UNIX workstations, interface using the X windowing system and Motif widgets.	IC fabrication, modeling deadlocks in FMS, modeling of placement machines for printed circuit-card assembly, and modeling of printed circuit-card assembly lines.
OSU-CIM	Smalltalk-80 on 486-based PCs, with interface using VisualWorks 2.0	Kitting operation in electronics manufacturing and evaluating decisions in operating FMS [6].
SmartSim/ SmarterSim	Smalltalk-80.	Small manufacturing cells, e.g., robot manufacturing cells.

Table 2.3. The author's research objectives and OOP rationale

Research Objectives	OOP Rationale
1. Developing a modeling environment that implements hierarchical modeling 2. Developing a basic element that is able to capture any process, therefore simplify the syntax and make it easy to learn and maintain. 3. Developing a framework that is able to implement distributed computing to make the system run faster. 4. Developing a system that is platform independent.	1. Reusability 2. Inheritance 3. Encapsulation 4. Modularity

Table 2.4. DOBIS's execution platform /develop environment and intended applications

Platform/Environment	Applications
Environment: Java Programming Language Platform: Any PC	Industrial and Manufacturing systems where time and capacity of the resource are the concerns.

To compare the author's work with existing researches, please see Table 2.3 and Table 2.4.

2.2 OBJECT-ORENTED PARADIGM AND JAVA PROGRAMMING LANGUAGE

The Object-Oriented paradigm has great impact on the modeling methodology of the simulation. This concept was first introduced in the SIMULA simulation system in the 70's, because it is intuitive to view the manufacturing systems as composed of interacting objects, e.g., the machines, the workers, the parts, the tools, and the conveyors. Also, the part routings, the schedule, as well as the work plan could be viewed as objects [79]. In the Object-Oriented paradigm, an object has *attributes* that represent the state of the object and functions to describe the ways upon which the object can be operated [78]. For example, a machining centre can have attributes like capacity, busy or idle, and functions like lathe, drill, etc. In Java and similar languages, the functions are implemented by *methods*. An object's attributes and functions are encapsulated within the object's construct through *encapsulation*. Objects interact through predefined interfaces so that information of an object will not spread all over the system thereby making it difficult to be modified [79].

Objects that have common structures and characters can be grouped into a *class* [36]. For example, all types of lathes can be grouped into the class lathe. Different *instances* of the classes are generated with different parameters. Therefore, a lathe class is able to have as many different instances of lathes as the users want. A lathe class can be extended to a new class with added functions through *inheritance*, e.g., a lathe with a special fixture.

The importance of inheritance is directly reflected in the model reusability. The reusability of the models has led to considerable flexibilities in modeling. It provides users a way to reuse some portion of the classes and can flexibly model complex manufacturing systems to some extend. Luna [12] has identified the benefit of hierarchical structure of classes in that a better structure will result in better reusability.

Another main objective of developing this simulation system is to provide the platform independent execution capability. Java meets this requirement, implements Object-Oriented paradigm and provides a friendly developing environment as well [78].

One of the concerns of employing Java instead of other Object-Oriented Programming languages is that other Object-Oriented programming languages, e.g., C++, provides Multiple Inheritance (MI), while Java does not. MI means one class can have multiple superclasses, while single inheritance can only have one. Although it is generally seen as a convenient instead of an essential feature [77], MI causes confusion when methods of the superclasses share the same name, and it makes the implementation complex [75, 76]. Java, on the other hand, supports single inheritance to avoid the complexities [75]. However, Java provides *Interface* that serves as a mechanism to emulate MI, which is discussed in

[75].

Up to now, the MI has been discussed as a feature of the programming language by the author. It is worthwhile to discuss it from the modeling point of view. Existing OO simulation systems provide predefined object structures and functions to model real world resources, e.g., a machining centre and an AGV are totally different. When there is a new resource that has both of their characteristics, a new class that inherits both machining centre and AGV classes is needed. Therefore, MI is important to the modeling abilities in existing OO simulation systems.

In the system proposed by the author, the real world objects are recursively decomposed according to the time an internal operation takes. In this light, a certain operation performed by a machine, at a certain level of detail, is a composition of other operations, e.g., (a) and (b) in Figure 2.15. Therefore, from a software engineering point of view, any operation can be an object or a container of objects. An object could be a whole production line in the large, or a tiny internal operation in the small. Therefore, because of hierarchical modeling, new objects are constructed through *composition* instead of inheritance. In other words, in DOBIS, the inheritance feature provided by the programming language benefits programmers who construct such a simulation system, instead of users who apply this system to build models. Therefore, MI is not a concern in modeling ability of DOBIS.

While both C++ and Java are Object-Oriented programming languages, Java provides platform independent execution ability that is important to the author's framework.

Therefore, the author chooses Java as the developing environment.

2.3 MODELING STRATEGIES AND CURRENT SIMULATION SYSTEMS

Traditional simulation systems and OO simulation systems are all based on one of the three generally accepted modeling strategies: Event-Scheduling, Activity-Scanning and Process-Interactions, to implement the modeling tasks [5, 6, 87]. Existing OO simulation systems take advantage of the Object-Oriented programming language to achieve higher levels of reusability. Section 2.3.1 reviews the three modeling strategies; section 2.3.2 reviews current modeling methodologies.

2.3.1 Modeling Strategies

The Event-Scheduling strategy is used to describe the actions the objects will take once an event is executed by the simulation system (Figure 2.3) [5, 6]. The event is drawn from the future event list generated by the objects. All the events in the future event list are ordered by their timestamp in a non-descending sequence. Once an event is executed, the virtual global time of the simulation system is updated to that time, the object in which the event occurs takes actions to change its own status and invokes other objects to change their status accordingly. Thus, in the Event-Scheduling strategy approach, the first step will be to interpret the industrial and manufacturing systems by identifying flow-entities and resources. Then a list of actions that *will* occur once an event is executed will be described individually according to the interactions between the entities and the resources. After the

simulation starts, the simulation system goes through the events picked from the future event list and invokes the corresponding event description. Since this strategy views and handles the resources in the manufacturing systems as individual objects, the author believes this strategy can be easily incorporated with the Object-Oriented paradigm. This strategy also provides the ability to hide the objects' internal operations from outside, leaving room and freedom for object-level modeling.

Arrival Event Description

1. Check the status of the ATM(s) (idle or busy)
   (a) If there is an idle machine
      (i) Start serving the customer, update idle status
      (ii) Generate a departure event for this customer
   (b) If all busy, place customer in queue
2. Generate new arrival event for next customer.

Departure Event Description

1. Customer Leaves
2. Check queue (empty or not)
   (a) If empty, set ATM to idle
   (b) If not empty then do
      (i) Choose a waiting customer
      (ii) Generate an arrival event for this customer

Figure 2.3. Arrival-Departure simulation for an ATM illustrated in Event-Scheduling strategy [6].

Since a resource changes its status only when it executes an event and it takes actions according to the *current* status of other resources, communications with other related resources is necessary after its own status has changed. An example is shown in Figure 2.4.

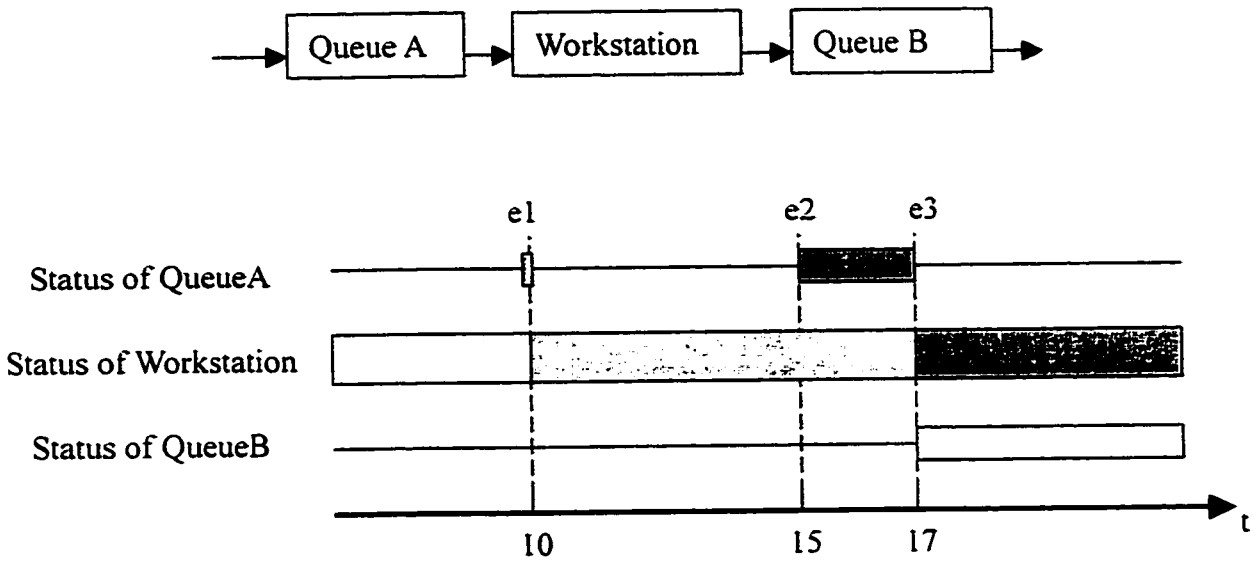


Figure 2.4. Event-Scheduling for a typical workstation cell

Rectangles in different shadings illustrate different workpieces; the length of the rectangle illustrates the duration that the workpiece resides in the resource.

Suppose at time 10, the first entity enters Queue A. Then Queue A identifies that the Workstation is idle, therefore the Queue A "pushes" the entity into Workstation and schedules the next event to be at time 15 to generate another entity. The "push" consists of two actions: first, the Queue A removes an entity from itself and inserts this entity into the Workstation; secondly, it sends a message to the Workstation to activate its functions in order to process the entity. The second action distinguishes the Event-Scheduling strategy

from Activity-Scanning strategy. In other words, in the Event-Scheduling strategy a resource has to activate other resources when its own status is changed, whereas in Activity-Scanning strategy a resource only takes care of its own changes. Suppose the Workstation schedules a processing time of 7 and put an event with timestamp 17 into the future event list. The next event is executed at time 15, therefore Queue A finds that the Workstation is busy so it keeps the new entity in its queue. Then at time 17, the Workstation finishes its work and "pushes" the entity into Queue B. But it is not enough to have "push" at this time. After the Workstation becomes idle, it has to send a message to Queue A so that it can get another entity and keeps working.

The Activity-Scanning strategy is similar to the Event-Scheduling strategy in terms of the form of event descriptions, except that it adds conditional actions (Figure 2.5) [5, 6]. In this strategy, once a resource changes the state of the system, the system scans all the conditions to see if any of the conditional actions can be executed. Compared to the Event-Scheduling strategy, in this strategy a resource does not have to activate others, the simulation system will do it. Therefore, the model is more independent. However, the cost of this feature is high. Because the system has to keep checking all the conditions, it imposes a great computation burden on the system and hence lowers the system's performance [53].

Event: an entity arrives at Queue A
Action: putting the entity into queue, scheduling next event

Event: Workstation current job completed
Action: pushing the entity to next queue, setting the Workstation status to idle

Event: Workstation start working
Condition #1:
a) Queue A is not empty
b) Workstation is idle
Action: start working, set the Workstation status to busy, schedule finishing time.
Condition #2:
a) Queue B is not full
b) Workstation is idle
Action: start working, setting the Workstation status to busy, schedule finishing time.

Figure 2.5. The example in Figure 2.4 illustrated in Activity-Scanning strategy.

A potential problem pointed out by Evans [5] is that there is a "fair" problem when the test head (see nomenclature) scans the conditional activities. Suppose there are two conditional activities which have condition case#1 and case#2 respectively (see Figure 2.5). The test head will always scan case#1 first, and case#2 will be invoked only if case#1 is not satisfied. Therefore, the user implies sequential relationship between conditional activities which may not always be the user's intention. Another problem pointed out by [49] is that possible modeling inaccuracies may occur with the activity-scanning strategy because discrete time slices must be specified. If the time interval is too wide, detail is lost.

Activity-Scanning strategy is capable of modeling such systems in which interactions

between independent components are complex. For example, the components act differently according to different situations [16, 38, 40]. The author believes that it is especially fit for the Agent-Based simulation systems. One key character of Agent-Based system is that the agent is autonomous in the system. The agents act without others controlling their actions and internal states [14]. This is not the case in industrial processes, where automation lines and robots are controlled directly either by humans or by program. Agent-Based systems are interested in investigating the behaviour of the agents and their interactions with their environments [15, 16]. The simulation systems for manufacturing systems are interested in the manufacturing systems' performances.

Process-Interaction strategy describes the system from the viewpoint of an entity flowing through the system (Figure 2.6). The model built under this strategy is the "life-history" of an entity [5, 6]. The process defines the entity's resource-requirements, possible impediments, and its duration in each of its activities. Hence the outline of the manufacturing system and the relationships between the resources are clearly defined. Instances in the system will flow in a parallel pattern and each of them will take a copy of the process information for its own flow. Although the process is described in a sequencing style, the running simulation contains many instances of the process existing at different stages of development, giving rise to the overall haphazard pattern of event occurrence [5].

```
GENERATE customer EVERY arrival
TIME goods_selection
ACQUIRE check-out_operator
TIME check_out
RELEASE check-out_operator
BULK customer
```

Figure 2.6. An example of process-interaction strategy [5]

Process-Interaction strategy models can be extremely compact, but they are incapable of modeling complex systems [53]. Macro and Setterdahl [3] pointed out that a flaw of the Process-Interaction based simulation systems was that the traditional system modeled real-world system according to the flow entities rather than the controller interactions that governed the flow entities. The flow entity interacts with the resources in the manufacturing systems sequentially. Inevitably the modeling ability of this strategy has a limit: processes in the manufacturing systems being studied have to be sequential. An example illustrating this limitation is given in [53]. In this example, a process needs two resources to work simultaneously. The program is shown in Figure 2.7.

According to [53], the Resource_Type_1 may be reserved well before the Resource_Type_2 becomes available. In the meantime, the Resource_Type_1 is prevented from performing useful work elsewhere. Another concern is that the user is using a sequential process to model a parallel process, which raises confusion in logic.

QUEUE(RESOURCE_TYPE_1)

SEIZE(RESOURCE_TYPE_1)

QUEUE(RESOURCE_TYPE_2)

SEIZE(RESOURCE_TYPE_2)

DELAY(OPTIME)

RELEASE(RESOURCE_TYPE_1)

RELEASE((RESOURCE_TYPE_2)

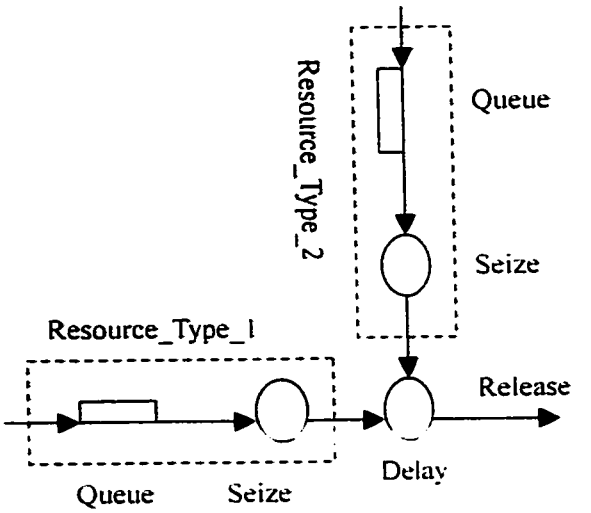


Figure 2.7. Two resources are needed simultaneously to work, illustrated in Process-Interaction strategy

There are several simulation packages that are widely used by simulation practitioners. GPSS is a high-level simulation language based on Process-Interaction strategy [5]. The most immediate impression of GPSS is its extreme brevity (Figure 2.8). GPSS gets straight on with the description of the entity process without the need of preliminary declarative segments. SIMAN as a Process-Interaction based simulation package has been proven to be powerful in the manufacturing domain [7]. SIMAN employs two descriptions to build one model. One is the experiment file that is for declaring the resources and their capacities. The other one is the model file used to describe the interactions between the entities and the resources based on the Process-Interaction strategy. The advantage of having separate files is that users are able to have several model files and one experiment file to test out different system configurations, e.g., different routing configurations.

```
* BARBERSHOP MODEL 1
        SIMULATE
* BLOCK DEFINITION CARDS
        GENERATE    18, 6    CUSTOMER ARRIVES
        QUEUE       LINE     JOINS WAITING LINE
        SEIZE       JOE      CAPTURES JOE
        DEPART      LINE     LEAVES WAITING LINE
        ADVANCE  16, 4    GETS HAIRCUT
        RELEASE     JOE      FREES JOE
        TERMINATE   1        LEAVES SHOP
* CONTROL CARDS
        START       25
        END
```

Figure 2.8. An example of GPSS simulation [62]

```
public void process ( )

{
  create ( expInterArrivalTime.sample( ) ); // create next Customer
  attArrivalTime = time;                    // record time in attribute
  queue ( queInputQueue );                  // insert Customer into queue

  // status delay: wait while the condition is true
  while ( condition( resServer.getAvailability( ) == 0 ) );
  seize( resServer );                       // decrease availability
  dequeue( queInputQueue );                 // remove Customer from queue
  obsTimeInQueue.record( time - attArrivalTime); // record status delay
  // time delay: service time
  delay ( expServiceTime.sample( ) );       // delay for service
  release ( resServer );                    // increase availability
  obsTimeInSystem.record( time - attArrivalTime); // record time in system
  dispose( );                               // recycle Customer object
 }
```

Figure 2.9. A simulation example from Silk by ThreadTec.Inc

The "//" tag indicates the text after it is comment.

Some newly developed Java based simulation systems, for example the well known

Silk and Simjava, are based on the Process-Interaction strategy [63]. The modeling information they provide is similar to the GPSS's or SIMAN's (Figure 2.9). The difference is that they benefit from the Java's Object-Oriented programming ability to construct reusable classes to model the resources of the manufacturing systems.

2.3.2 Current Modeling Methodologies

Hierarchical Modeling is employed to model complex systems and objects because it is able to recursively decompose objects into desired level of detail and structure the model information [57, 61, 67]. The *Discrete Event System Specification* (DEVS) [6,32,33,51,80,81] is a system theoretic formalism for modular, hierarchical discrete event modeling and simulation, it allows a modeller to specify a hierarchically decomposable system as a discrete event model that can be later simulated by a simulation engine. Sargent [61] lists the advantages of hierarchical modeling: reduction in model development time, support for reuse of a database of models, and aid in model verification and validation.

The DEVS is extended to Parallel DEVS to address collision issue and facilitate distributed and parallel computing in [81]. A DEVS model can be either an *atomic* model or a *coupled* model. The *atomic* class realizes the basic level of the DEVS formalism with specifications of the elemental components of the system of interest. The *coupled* class realizes the compositional constructs for structuring DEVS models into system hierarchies. A DEVS atomic model specification defines the states (variable values) and associated

time bases resulting in piecewise constant trajectories (see glossary) over variable periods of time, see Figure 2.10.

$M = < X_M, Y_M, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$

where

$X_M = \{(p,v) \mid p \in IPorts, v \in X_p\}$ is a set of input ports and values

$Y_M = \{(p,v) \mid p \in OPorts, v \in X_p\}$ is a set of output ports and values

S is a set of states

δ_{int}: $S \rightarrow S$ is the *internal transition* function

δ_{ext}: $Q \times X_M^b \rightarrow S$ is the *external transition* function

δ_{con}: $Q \times X_M^b \rightarrow S$ is the *confluent transition* function

λ: $S \rightarrow Y_M^b$ is the *output* function

ta: $S \rightarrow \Re_{0,\infty}^+$ is the *time advance* function.

with

$Q = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$ is the set of *total states*

e is the *time elapsed* since last transition

X_M^b is a bag of inputs (a set with possible multiple occurrences)

Y_M^b is a bag of outputs (a set with possible multiple occurrences)

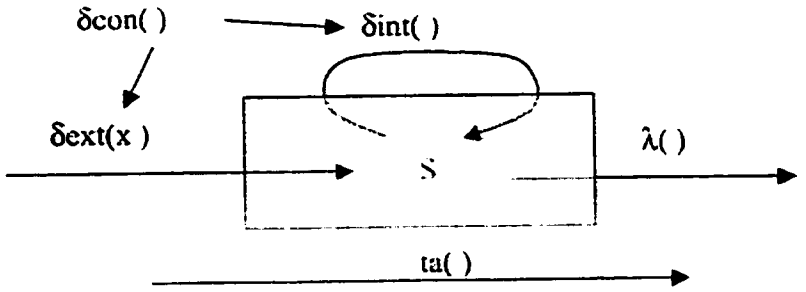


Figure 2.10. Atomic Model of Parallel DEVS

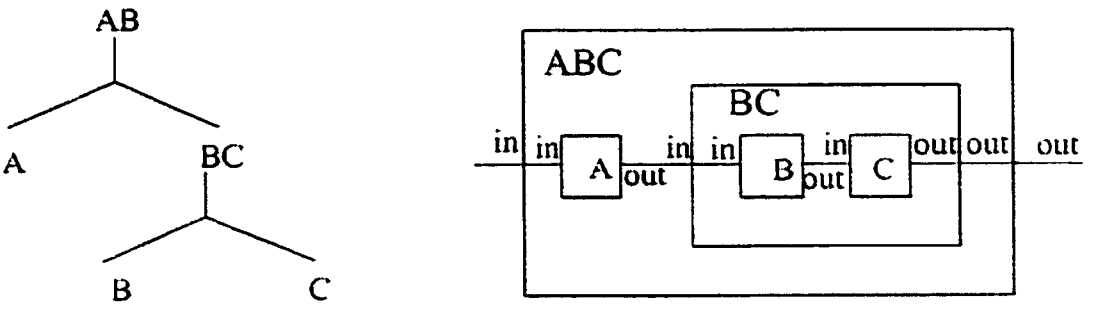
As an example given in [33], when an input packet arrives at the queueing system, the instance variable S which is the length of the queue should be incremented by 1. The function which carries the input and then changes the instance variable is the δ_{ext} *function. After an elapsed time given by the ta function, the system checks the queue, it removes some packets causing the length of the queue to decrease by the amount of the removed packets. In other words, the instance variable, the length of the queue, changes internally from the previous state to the next state which is less by the number of the removed packets than the previous one. Likewise, the internal function* δ_{int} *change internal variables from previous state to the next. The* δ_{con} *function decides what to do when both external and internal events occur together. The* λ *function produces output Y from the instance variables. ta returns the time to the next internal event.*

A DEVS coupled model designates how (less complex) systems can be coupled together and how they interact with each other, see Figure 2.11.

$DN = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$
Where
X is a set of input values
Y is a set of output values
D is a set of the DEVS component names.
For each $i \in D$,
M_i is a DEVS component model
I_i is the set of influencees for i.
For each $j \in I_i$,
$Z_{i,j}$ is the i-to-j output translation function.

Figure 2.11. Coupled Model of Parallel DEVS

An example of coupled model is illustrated in Figure 2.12.



Hierarchical Tree Coupling Relation

Figure 2.12. Coupled model ABC [33]

BC is a coupled model of B and C; ABC is a coupled model of A and BC

According to [32], a formalism is said to be closed under composition if any composition system obtained by coupling components specified by the formalism is itself specified by the formalism. The closure of a formalism under composition is required in

order for the formalism to lend itself to hierarchical model specification [32]. The closure of the Parallel DEVS is demonstrated and proved in [81].

The Hierarchical Control Flow Graph (HCFG) [57-61] applies hierarchical modeling based on the Control Flow Graph. In the HCFG approach, models consist of a set of independent, encapsulated, concurrently operating model components where each component has its own thread of control and the components interact with each other solely via message passing. The building block of the models is the Model Component. The HCFG models have two complementary types of model information: one is called Hierarchical Interconnection Graph (HIG), which is used to specify the set of encapsulated components that comprise the model and how those components are interconnected, illustrated in Figure 2.13; the other one is the hierarchical control flow graph that is used to specify the behaviours of the individual model components, illustrated in Figure 2.14. The HCFG approach inherits the encapsulation concept from Object-Oriented paradigm. This concept is implemented by the models' external view and internal view, which is illustrated by Figure 2.15.

The hierarchical structure is able to support the development of hierarchical models using "top down" recursive decomposition of components into smaller and simpler components, and using "bottom up" composition of existing components to form new components [59]. Both DEVS and HCFG lack the ability to explicitly represent the logical relationships between the internal operations, e.g., the two (A) components in Figure 2.15. The DEVS approach and the HCFG approach are well suited for the Object-Oriented

paradigm, because they all facilitate similar concepts such as encapsulation and inheritance.

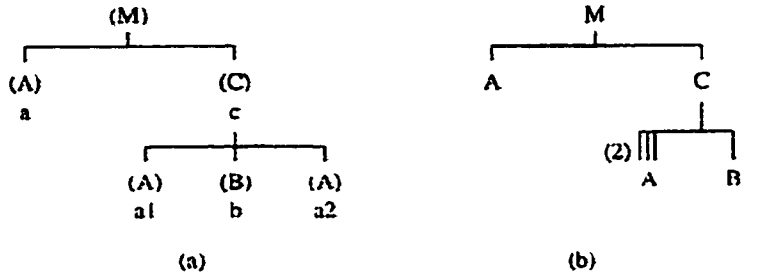


Figure 2.13. Hierarchical Interconnection Graph [59]

In this graph, component M consists two types of sub-components A and C. C consists two types of sub-components A and B.

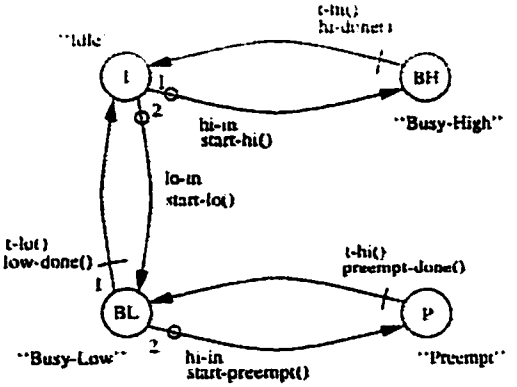


Figure 2.14. Control Flow Graph [59]

Petri Nets is used to model the manufacturing systems by many researchers [31,69-74]. The basic Petri Nets employs four elements, which are token, place, transition and arc, as illustrated in Figure 2.16.

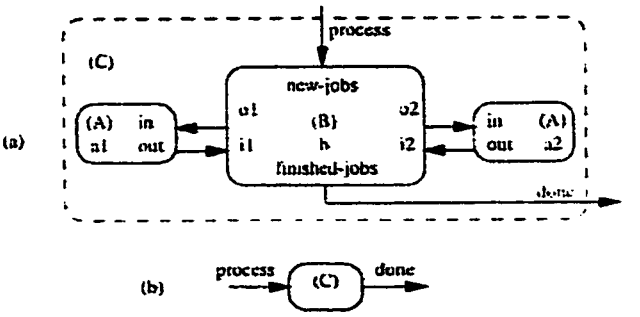


Figure 2.15. HCFG Model example [59]

(a) is the internal view that represents the detail of the object-level model information, components are connected by their interfaces. (b) is the external view of the object.

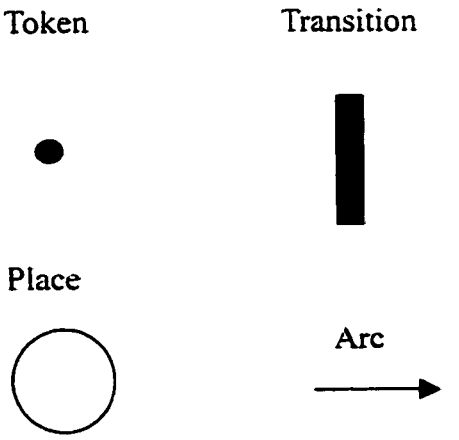
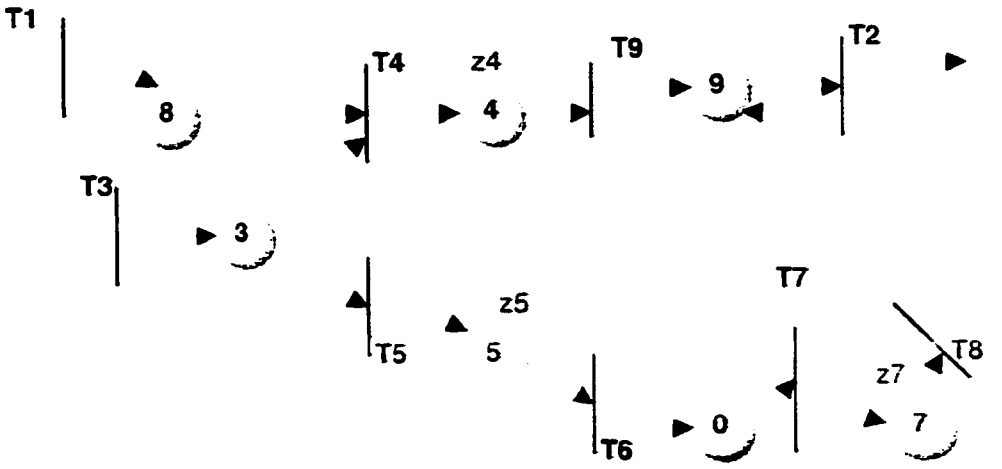


Figure 2.16. Token, Place, Transition and Arc

In the manufacturing domain, a transition can be interpreted as a processor; a place can be interpreted as a buffer or a condition; an arc can be interpreted as a logical relationship between transitions and places. A Petri Nets is marked with tokens in places to illustrate its state. The basic Petri Nets does not distinguish tokens, nor does it capture time of processes. This apparently limits the ability to model complex system. Therefore, several extensions

of Petri Nets have been developed [69, 71, 73], e.g., Coloured Petri Nets, Timed Petri Nets, Object Oriented Petri Nets, etc. These extensions make it possible to capture time and capacity constraint for resources in the manufacturing systems domain, and distinguish the tokens that usually denote parts. Petri Nets is able to model concurrent processes, its graphical representation makes it possible to visualize the state-flow of a system and to quickly see dependencies of one part of a system on another. Because of this, Petri Nets is often used to model the control logics of manufacturing systems. For example, Barad [69] discussed modeling the control of AGVs by Timed Petri Nets as illustrated in Figure 2.17.



T1 - call arrival for new pick-up
T2 - start new loaded travel
T3 - last loaded travel completed
T4, T7 - start moving to new pick-up
T5 - start moving towards staging
T6 - staging reached
T8, T9 - new pick-up reached

P8 - call is waiting for AGV
P3 - AGV free at last drop-off location
P4 - empty travel from last drop-off to new pick-up
P5 - empty travel from last drop-off to staging
P0 - AGV at staging
P7 - empty travel from staging to new pick-up
P9 - AGV ready to start loaded travel

Figure 2.17. Control of AGV modeled in Petri Nets [69]

However, because the Petri Nets graphic representation is static, e.g., a place or a transition remains unchanged when a resource changes from idle to busy, it has to apply additional nets to model the status of the resource. Therefore, control flow, information flow and the material flow have to be modeled separately but within the same net. This causes confusion, e.g., Figure 2.18 illustrates a Petri Nets model for a machining centre. This problem will be serious when the model is complex.

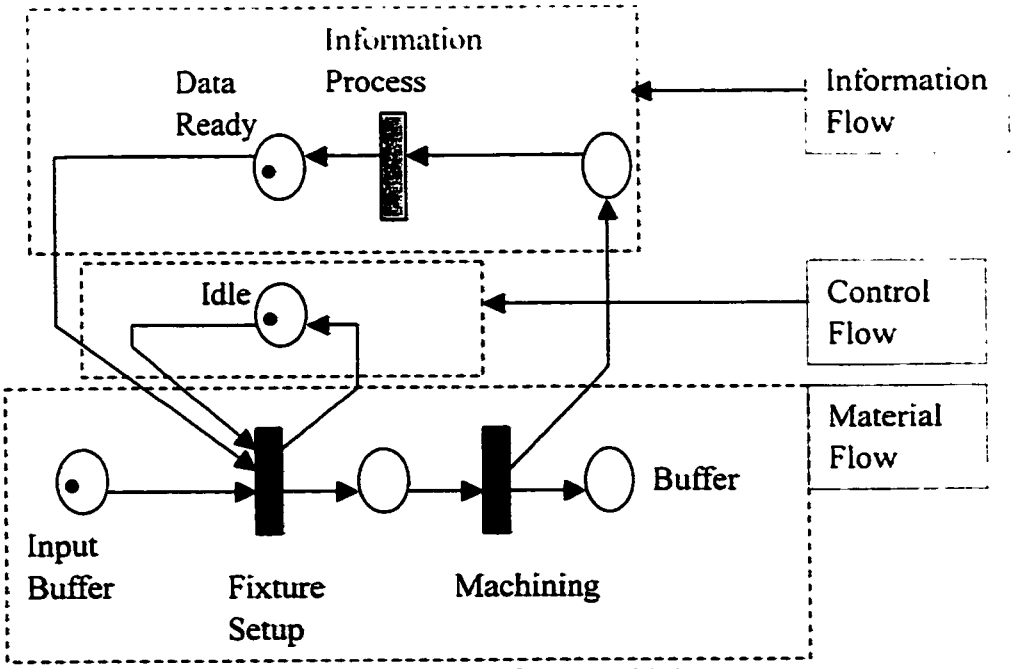


Figure 2.18. Petri Nets model for a machining centre

Three kinds of flows: material flow, control flow and information flow identified by dash line boundary. The status of the system is represented by the status of the three flows.

The logical relationships between processes are implicit in Petri Nets models, e.g., the relationship between Input Buffer, Idle and Data Ready in Figure 2.18, and the relationship

between T4, T5 and place 3 in Figure 2.17. The Petri Nets approach demonstrates that a manufacturing system can be modeled by composing basic elements: transition and place. And a process can be modeled by capacity and time. Furthermore, although it is not clearly indicated, a place or a transition can be further decomposed into a subnet, and hence to achieve recursive decomposition ability.

Fishwick discussed multimodeling approach in [57, 68]. Multimodeling, in short, is to apply different approaches to model real world system in the same model structure, because each approach can only answer limited types of questions. The purpose of developing such approach is to enable the system to answer more different kinds of questions. To illustrate, Fishwick gave an example in [68]. In this example, a pot of boiling water is modeled by Continous State Space, Petri Net and Finite State Automata. It is able to answer a variety of questions, e.g., "What can happen immediately after the pot boils?", "How long does it take for the water to cool to room temperature if the knob is turned off when the water is 90 degrees?". But this approach does not address the concerns about the flexibility in terms of resolution. Furthermore, such a system certainly needs the users to be specialists and leads to higher building and maintaining costs than other systems.

Besides the approaches applied, objects and functions that are abstracted from the real world and provided by the simulation systems have great impact on the modeling ability. Existing simulation systems provide predefined functions to model resources in the manufacturing systems. This is due to the intention of hiding the general purpose programming language behind the systems, so that users do not have to spend time and

effort to master the programming language that is irrelevant to the application. However, since traditional simulation systems do not provide model reusability, these systems are sufficient only for simple, one-time modeling type of applications in manufacturing [2]. Unlike traditional simulation systems, OO simulation systems provide a better way for users to customize the model of the resources, giving users more freedom to solve problems in various situations [2, 61, 66]. Because of the OO paradigm's native inheritance and encapsulation ability, systems using this paradigm usually achieve high reusability [2-4, 8-13, 61, 66]. However, both traditional and OO simulation systems employ dozens of commands in their modeling environment, e.g., the current number of the commands in the SIMAN simulator is 64 pages [33], which down grades the flexibilities and ease-of-use abilities, and impose longer learning period, higher maintaining cost and more chances of making mistakes. Furthermore, limitation on the practical number of commands and parameter formats constrains the expression of complex behaviors, and makes the simulators incapable of extending the problem space or describing models deeper than some fixed level.

2.4 THE DISTRIBUTED COMPUTING ALGORITHM

In this section, the author identifies the definition of distributed simulation and reviews a commonly accepted distributed computing paradigm.

2.4.1 Definition of Distributed Simulation

There is no generally accepted definition of distributed computing. In general, computing means not only calculating, but also anything relates to computer operations, e.g., data storage, information searching, etc. In [18, 19], this concept is defined as the remote execution of simulation software under a web browser or construction of simulation models, so that simulation experts do not have to be collocated. Page [20] tried to conclude the application of distributed computing into four categories, which are: Simulation as hypermedia, Simulation research methodology, Web-based access to simulation programs, Distributed modeling and simulation, Simulation of the WWW.

The author of this proposal takes the definition used in [9, 21, 26], which define the concept as to partition the simulation task into smaller processes, distribute these processes to the computers over the network and execute them simultaneously so that the overall simulation executing time is shortened.

In distributed computing, CPUs do not share the same memory and they are separated over the network and could be in different geographical locations. They connect to each other through network protocols [21].

2.4.2 Distributed Simulation Paradigms

Ferscha and Tripathi [22] proposed four levels of distribution. The application-level is to run the replications of the same simulation model simultaneously on different computer.

The subroutine-level is used in the case where the input parameters of replication(i) are determined by the results of replication(i-1). In component-level, the objects of the resource-limited model are assigned to different computers. At event-level, events of simulation model will be distributed among processors for concurrent execution.

Since this proposal is primarily concerned about simulation application in engineering domain, the development of distributed computing algorithms is purely in the domain of Computer Science and is out of this proposal's scope. The author's goal is to utilize one existing algorithm in the proposed framework to prove it is possible for the proposed simulation system to utilize network-based computing power for execution.

Generally, there are two steps to go through when designing distributed systems [21]. The first step is to partition the model into small sub-models, this is often referred to as partitioning. The second step is to determine what kind of protocol is going to be applied, either conservative or optimistic.

Partitioning is a process that divides a simulation model into small sub-models. Usually each sub-model is referred to as Logical Process (LP). Each LP should reside on a computer and several computers execute the simulation simultaneously. Each of the LPs has a local virtual time (LVT), which advances the time during the simulation execution. LPs communicate with each other through message-passing with time stamps to synchronize their local clocks [23]. One concern about distributed simulation is the local causality problem [21, 22, 24]. A local causality is violated when a LP receives a message with a timestamp lower than its local time [21].

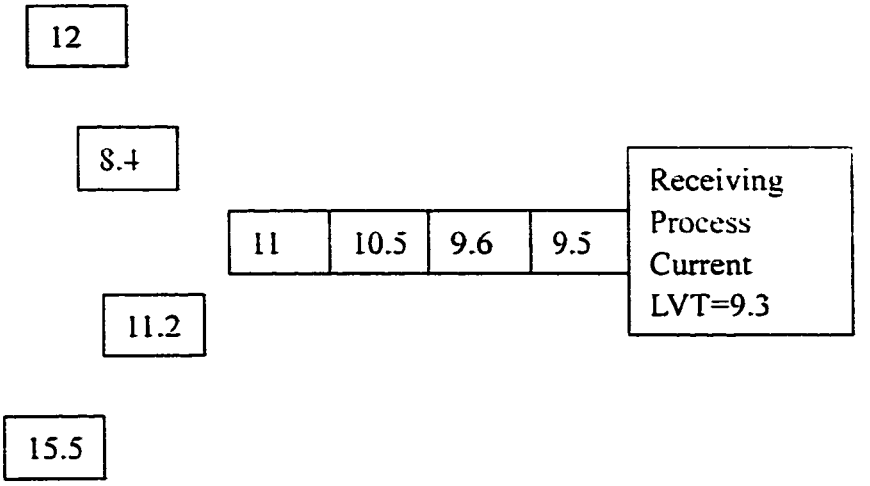


Figure 2.19. An illustration of local causality problem

For example in Figure 2.19, a message with a timestamp of 8.4 is sent to the receiving process of a LP. The current LVT of that LP is higher than the timestamp hence a causality violation problem is generated.

There are two approaches to synchronize the local clocks: Optimistic and Conservative. Fujimoto made an extensive discussion about these two approaches in [85]. In the conservative strategy, the time order of events is strictly guaranteed by preventing the simulation engine from processing an event that has causal dependencies pending. In optimistic strategy, the simulation is rolled back only if premature processing of local events is inconsistent with causality conditions produced by other processors. The most widely used optimistic approach is the Time Warp algorithm (TW). The TW assumes that any moment the messages that are already in the queue are in none descending order according to their timestamp. New messages can arrive asynchronously during the execution.

There will not be a problem as long as the messages are arriving with timestamp higher than the LVT. But whenever a message arrives with a timestamp *t* less than the LVT, the Time Warp (TW) must

a) roll back the process to a time just before the t

b) execute the new message at virtual time t

c) start re-executing messages with timestamps greater than t in none-descending timestamp order, cancelling all of the effects of any output messages that were sent after t [23].

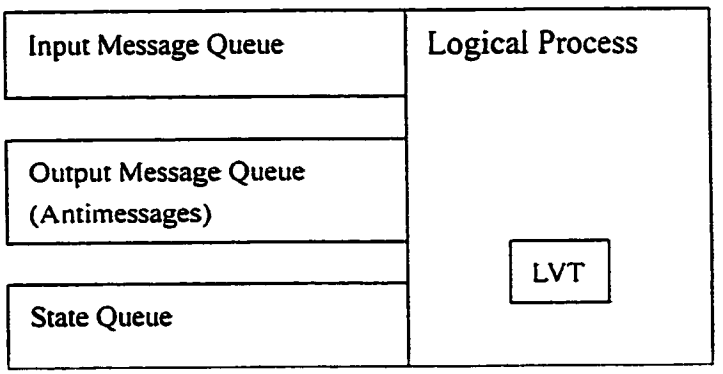


Figure 2.20. Logical process embedded with Time-Warp algorithm

In order to support rollback, the TW has to regularly save the state of each process (Figure 2.20). These states are stored in a queue associated with the process and are invoked whenever a roll back happens. In order to implement the step c), TW introduces the antimessage. In TW, every message exchange among LPs is considered to have a sign, with either + or -. Whenever a LP requests a message to be sent, TW creates a message-antimessage pair. The positive message will be put into the destination's input

message queue and waits to be processed. The negative message remains in the output message queue of the LP. If there is no roll back during execution, the negative message remains in the queue and is finally collected and destroyed. If a LP rolls back to simulation time t, the simulation will take a different path than before. TW compares every message-send request from LP with the old negative messages in LP's output queue. If a new message is already represented in the output queue, both this message and its antimessage are discarded because the receiver has already got a copy. This algorithm has been proven to be able to implement distributed discrete event simulation [21, 22, 25-27].

There are several other approaches proposed. For example, an optimized Time Warp approach was proposed in [86] to avoid chain roll back. A chain roll back takes place when a receiver of anti-message (LP) rolls back and sends out more anti-messages, resulting in a chain reaction within the system. This will dramatically increase the communication load of the system and downgrade its performance. An adaptive protocol was brought out in [17]. The adaptive protocol, simply put, is a combination of conservative and optimistic approaches. According to [17], the conservative approach tends to do poorly when job number is small, and much better as the number of job increases; on the contrary, the optimistic approach tends to do better when the number of jobs is small, and its performance becomes worse when the number of jobs increases. Since it is only for demonstration purpose, the author is not going to discuss distributed computing in any detail. Instead, the author is going to take one approach, the Time Warp approach, to show the possibility of applying distributed computing to DOBIS.

CHAPTER THREE

A NEW FRAMEWORK FOR AN DISTRIBUTED OBJECT-BASED INDUSTRIAL SIMULATION SYSTEM

3.1 GENERAL APPROACH

In Chapter 2, the author reviewed the modeling strategies and methodologies for industrial and manufacturing systems. The Parallel DEVS provides a formal theoretical foundation for hierarchical modeling of discrete event systems, and it is well suited for the Object-Oriented paradigm. However, the logical relationships between the internal operations are not considered in DEVS formalism.

The Event-Scheduling strategy has the desired independent modeling ability, and provides similar concepts to the Object-Oriented paradigm such as encapsulation. Therefore, it is possible to combine Object-Oriented technique with this strategy to model system-level information and to provide a platform for object-level modeling.

In this chapter, the author first informally presents a strategy to implement system-level modeling, then develops the Proto-Element (PE) and the Coupled PE (CPE) to model real world systems to implement object-level modeling. Then, the author maps the PE and CPE into an extended Parallel DEVS to provide formal definitions. Finally, the author abstracts a set of classes and methods to demonstrate the implementation of the PE and CPE. The OO technique is the foundation of the proposed framework. Its important

features like inheritance and encapsulation are key features to the new framework and have been widely recognized [1-4, 9-13, 28-30] as relevant in any modeling domain.

Finally, the author presents a straightforward algorithm based on component-level distributed simulation to illustrate the possibility of distributing the system.

3.2 The New Framework

The author views the modeling of industrial and manufacturing systems as having two levels, the system level and resource level. According to the OO concept, the resource level is called the Object level. At the system level, the routing and resource-interaction information is described; internal operations, their parameters and logical relationships are defined at object level.

3.2.1 System-Level Modeling and Node-Based Modeling Strategy

Because of the independence of the modeling information and the similarities to the Object-Oriented concept, the author concludes that the Event-Scheduling strategy is more capable of modeling complex manufacturing systems.

The proposed strategy for system-level modeling is called Node-Based strategy (Figure 3.1). In this strategy, the resources are treated as objects, and these objects are seen as nodes in the system. Therefore, the objects interconnect each other by their interfaces.

and their internal operations are hidden from the system level. Research in the DEVS and the HCFG has shown that uniform interfaces are essential to hierarchical modeling ability. The interfaces of the Node are the two actions, "pull" and "push" as discussed in Chapter Two, to perform when an event is executed. The *pull* action is the action of the object getting an entity from its preceding nodes; whereas the *push* action is the action of the object sending an entity to its following nodes. As interface, the "pull" only connects to "push" so that the direction of the entity flow is consistent. The interface also keeps the information of the nodes' preceding and following nodes for the two actions. Therefore, the routing information is kept in the objects. This is the basic notion underlying the Node-Based strategy.

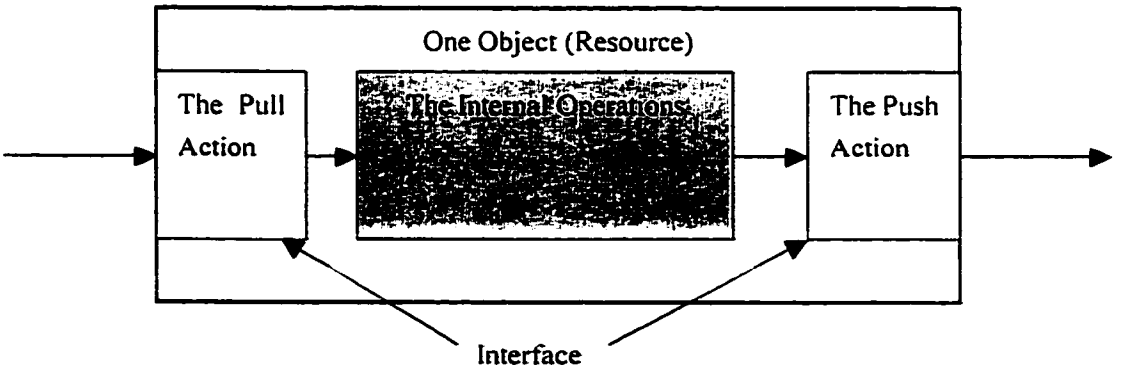


Figure 3.1. The structure of Node

Internal operations are encapsulated within the node; nodes are connected by their uniform interfaces.

Similar to Petri Nets approach, which models the systems by *places* and *transitions*, the Node-Based strategy views the industrial and manufacturing systems as having two kinds of resources: Workstation and Queue. This is enough to model the routing

information and locations where the entities are physically processed. In order to distinguish the resources among the nodes, here are two definitions:

Definition: A workstation node represents any station that changes the entity's physical properties.

Definition: A queue node represents a place or a process where entities reside and does not cause physical changes.

According to the definitions above, a workstation node represents any machining centre, driller, surfacer, etc. A queue node represents queues, AGVs (Automatic Guided Vehicle), conveyers, storages, etc. Since industrial and manufacturing systems are resource-limited systems, there will be a starve/block problem when two workstations are directly connected. A starve problem occurs when in two stations (Figure 3.2) *A* and *B*, *B*'s processing time is shorter than *A*'s. When *B* finishes its work and becomes idle, it has to wait doing nothing until workstation *A* finishes its work and passes the workpiece to *B*. A block problem happens when *B* processes slower than *A*. In this case, when workstation *A* finishes its work and becomes idle, it cannot pass the part to *B*. *B* blocks other parts to be processed by *A*.

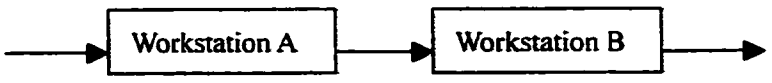


Figure 3.2. An illustration of starve/block problem

Now, the question is how the nodes should be connected. In reality, any workstation or queue must have some node before and after it. Obviously, it will be redundant if users assign both preceding and following nodes. The question is should users assign the flow entities where to go or where from. When building a complex model from scratch, it might be hard to know the detail of the next node, for example the ID, the input, etc. That means if the user has to assign the node's following node, he has to look for something that has not been built. On the other hand, if the user assigns the node's preceding node, he can have a clear view about what has been done and what needs to be done in the current node. This gives the users better view and control over the constructions of the nodes and the model.

Axiom: A node is determined by its preceding nodes.

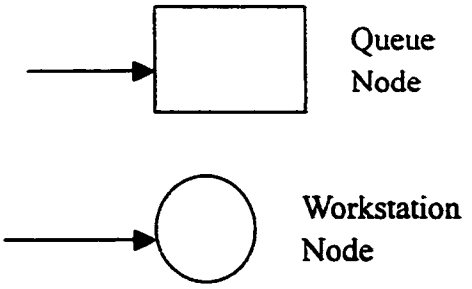


Figure 3.3. Queue node and Workstation node illustrated in graph form

Although a uniform structure is desired when the modeling task is carried out, graphical illustrations can help users to have an instant view of the whole structure of the system. So, in order to distinguish the Workstation Node and Queue Node in the graphical illustration, the author develops the icons shown in Figure 3.3.

A node should carry routing information, as defined in the following

Definition: Node = [ID, FromID [m], ToID[n], A] where:

a) ID: is the name of the node used to identify the current node.

b) FromID: the list of IDs of nodes from which the flow entities come

c) ToID: the list of IDs of the following nodes where the flow entities will be sent from the current node such that:

$$ToID_{preceding} = ID_{thisnode}$$

d) A: a finite set of attributes describing the properties and the status of the current node.

At system-level modeling, Node-Based strategy is applied to translate the real world manufacturing system into a network connecting each of the resources so that routing of entities and manufacturing resources can be easily controlled and modified. It also unifies the representation of the resources by ignoring their internal operations. To demonstrate its flexibility, let us take a look at the layout in Figure 3.4.

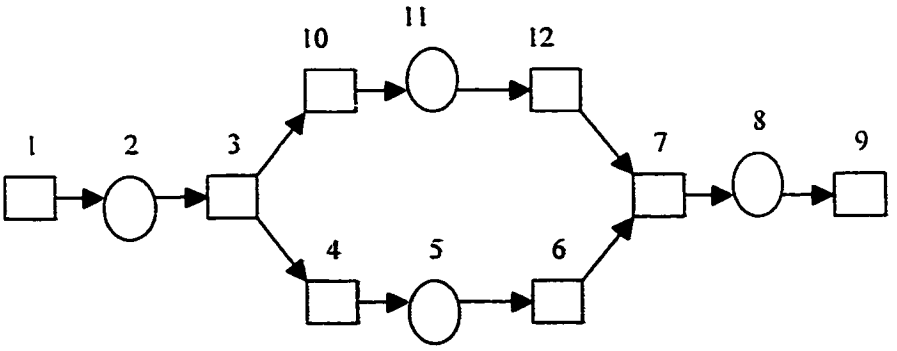


Figure 3.4. An example of Node-Based strategy

Suppose the original system has nodes 1 to 9. After running the simulation and analyzing the system's performance, it is desired to add another set of machines to the system as shown, node 10,11 and 12.

For demonstration purpose, the author defines the following notions:

a) *[a] From [b]*, *a* is the ID of current node, *b* is the set of IDs of the nodes where *a* pulls entities from.

b) *Generator: [c]*, *c* is the set of IDs of nodes that generate flow entities.

c) *End*, is the tag indicates the end of the codes.

Therefore, the pseudocode for modeling the original system with node from 1 to 9 is:

```
Generator: Node 1
Node 2 From Node 1
Node 3 From Node 2
Node 4 From Node 3
(to be continued)
Node 5 From Node 4
Node 6 From Node 5
Node 7 From Node 6
Node 8 From Node 7
Node 9 From Node 8
End
```

To add the three new node at the middle of the system, the user does not have to go back to check the codes, instead he can just add it at any point of the model, for example, the end.

Generator: Node1
Node 2 From Node 1
Node 3 From Node 2
Node 4 From Node 3
Node 5 From Node 4
Node 6 From Node 5
Node 7 From Node 6
Node 8 From Node 7
Node 9 From Node 8
Node 10 From Node 3
Node 11 From Node 10
Node 12 From Node 11
Node 7 From Node 12
End

When there are hundreds of lines of routing information, it will be hard for user to have a global view of the system by just a glance at the texts. So a graphical interface that implements the same semantics as indicated here for system-level modeling is desired in the future work.

3.2.2 Object-Level Modeling and the Proto-Element

While there are all kinds of ways that can be used to streamline the implementation and improve performance, such issues belong to the Computer Science field of study. The author is focusing on the usability/functionality aspects of the system. The implementation introduced and discussed in this thesis is a 'proof of concept' instead of 'production' version.

Before the author proceeds to further discussion of the object-level modeling, it is

important to identify the purpose of the object-level modeling. According to the definition given by the author, in the industrial and manufacturing systems domain, all resources have internal operations. The internal operations are embodied by parameters and control logics. In this context, the data is used to model performance information that is represented by parameters; the control logics are used to model control information that is represented by rules. From the software engineering point of view, the internal operations are storages for the parameters and the rules. The higher the resolution, the more internal operations emerge, and therefore more complicated logical relationships are generated. For example, a low-resolution model may be modeled by one internal operation; high-resolution models may have several parallel and sequential internal operations. To summarize the discussion above, the logical relationships form the framework of the model information, while the internal operations implement the framework with the parameters and the rules, and they all have to be captured.

The internal operation becomes the building block of the object-level modeling. The Node-Based strategy can be used in the system-level modeling because the resources are its building blocks. A resource is also a system that consists of many internal operations as its building blocks, and hence the internal operations are systems too. In the discussion of the example in Figure 1.1, the author proposed that internal operations, resources, and systems are all just different views of the same thing, and they can be modeled by a basic element.

In Petri Nets approach, resources in the manufacturing domain are modeled by two

basic elements: place and transition. However, the definitions of place and transition are implicit, e.g., a place can be a queue, it can also be a condition; a transition can be a machining centre; a conveyor does not change the physical property of a workpiece but it has to be modeled as a transition. From the definition of the internal operation, only the operations that have impact on the system's performance are modeled. However, when resolution is increased, a queue node at system level may have internal operations that have to be modeled at object-level modeling; a workstation node may have queue nodes at its object-level model. Therefore, the function of a node or an internal operation is dynamic. As an internal operation is decomposed, both queue nodes and workstation nodes may emerge. Therefore, it is necessary to develop a basic element that combines place and transition concepts. The only way to distinguish queue and workstation is whether they influence the system's performance, which can be measured by time.

The Petri Nets approach demonstrates that the performance of a process can be modeled by two parameters: processing time and capacity. Besides these performance parameters, implementation of control logics should be discussed as part of the basic element. Modern facilities, especially those computer controlled automated facilities, are relying more and more on the control systems to achieve high levels of flexibility [54]. Simple control logics can be FIFO/FILO (first-in-first-out/first-in-last-out) for storage equipments such as queues; more complex logics can perform different processes for different types of workpieces within a machining centre.

Pragmatically, an operation is realized by hardware, which is a place where

workpieces reside. Therefore, the basic element must have a mechanism to store the workpiece, where it is a form of storage. There are two kinds of storages that should be distinguished. For a workstation, machining time represents the time that a workpiece stays in the storage and there is usually no queueing rule; for a queue, which can be a special form of storage, time is not measured. Then, the illustration of the structure of the basic element is shown in Figure 3.5, it implements both place and transition functions in Petri Nets.

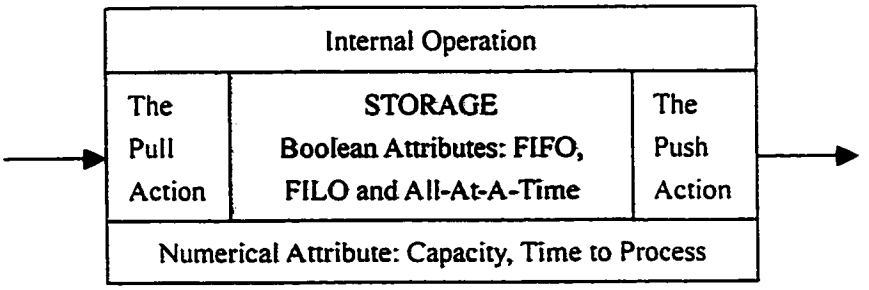


Figure 3.5. The structure of the Internal Operation

In order to inherit the graphical representation ability of the Petri Nets approach, the author developed the following graph for object-level modeling (Figure 3.6):

Figure 3.6. Internal Operation graph representation for object-level modeling

The length of the rectangle illustrates the time, relative to other internal operations, to process. It has functions of both place and transition as defined in Petri Nets.

The structure of the basic element is based on the structure of the node. The

Node-Based strategy is extended to the object-level modeling, and therefore the internal operations become nodes to the objects. The author moves one step forward, defines this basic element as the Proto-Element (PE), the basic element of both the object and the system. Hence, a uniform structure through out the simulation system can be achieved. The result is that the PE has such a modeling ability that it is able to capture any tiny internal operation, yet it can be a node at system level.

The working procedure of the PE is as follows, described in event description format:

Event: entity arrival

Actions:

1) Put entity into queue
2) Check capacity, if capacity is reached
    a) Set Boolean value Available = false
3) Schedule processing time, put into event list

Event: process finished

Actions:

1) Check if next element is available, if it is
    a) Push the entity to the next element.
    b) Set Available = true
2) Check if the preceding element is empty, if not
    a) Pull the entity from the precedent element to this element
    b) Start the actions for entity arrival

So, according to the author's extended definition, the PE consists of other attributes, for example, mean time between failures (MTBF), mean time to repair (MTTR), and status attributes, e.g., busy, idle, full, empty, system up or down. This is true in reality, where complex machining centres have several components and they are all internal operations of the machining centre.

Several PEs are connected according to their logical relationships to form complex resources. For example, in many cases, internal operations within one machine do not happen sequentially. Instead, they happen in a parallel manner, which means some operations are carried out simultaneously. It is necessary to add parallel notion into the PE structure.

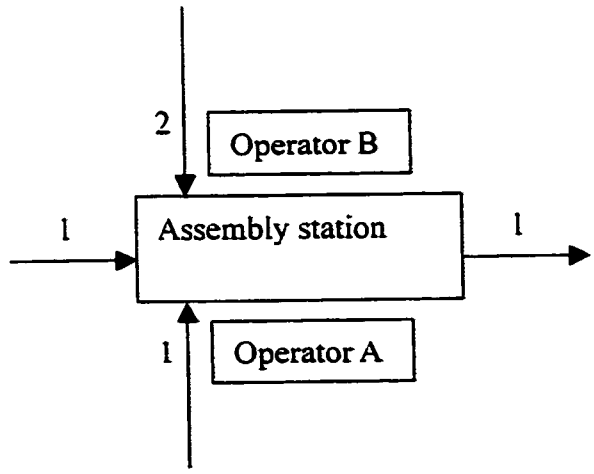


Figure 3.7. An example of two-operator assembly station

Arrows are directions of the material flows; numbers beside the arrows are capacities

Consider an example in Figure 3.7. In this workstation, two operators work together simultaneously (parallel). The time for them to finish their work is different. The work is

done only when both of them finish their tasks.

The following informal definitions are used to illustrates the notions of parallel PEs and assemble/disassemble processes:

Definition: PARALLEL {

PE1 { performance }

PE2 { performance }

...

PEn { performance }

}

Where, *PEn* is the IDs of the PEs that happen in parallel manner.

Definition: In the industrial and manufacturing domain. *assemble* is to bring two or more entities together to form a new entity; *disassemble* is to make one entity become two or more entities and apart from each other.

Therefore, a process like welding is also an assemble operation. The assembly station in Figure 3.7 can be modeled as following, texts after the tags "//" are comments:

Assembly_Station

PARALLEL {

```
Operator A {
    Performance. time ==4 //mean time for operator A to finish the work
    }
Operator B {
    Performance. time ==3 //mean time for operator B to finish the work
    }
;  this tag tells simulation engine the parallel part ends here
ASSEMBLE
```

However, this is still not sound in terms of logic. While we are clear about the logical relationship between the two PEs, the computer is not. A mechanism capturing the real world logic is needed. There are three basic logics, which are represented by three logical operators – *AND, OR* and *Mutually Exclusive OR (XOR)*. Their explanations from Merriam-Webster's dictionary are as following:

d) AND: a logical operator that requires both of two inputs be present or two conditions to be met for an output to be made or a statement to be executed.

e) OR: a logical operator that requires either of two inputs to be present or conditions to be met for an output to be made or a statement to be executed.

f) XOR: being related such that each excludes or precludes the other.

The *XOR* logic operator can also be explained as one and only one of two inputs must be true so that an output can be made or a statement can be executed. So three logic functions are put into the Proto-Element.

Beside the logic operators, there are two scenarios should be considered: 1) two PEs are two processes performed by the same resource, if one of the PEs is not free, then all the PEs are not free. In other words, the state of one PE (busy or idle) depends on the state of other's, the author defines it as *tightly coupled*; 2) two PEs are two resources, the state of one PE does not depend on other PEs, the author defines it as *loosely coupled*.

Therefore, the example in Figure 3.7 can be modeled as following:

```
Assembly_Station(tightly_coupled)
    PARALLEL_AND {
        Operator A {
            Performance. time ==4 //mean time for operator A to finish the work
            }
        Operator B {
            Performance. time ==3 //mean time for operator B to finish the work
            }   }
    ASSEMBLE
```

The author develops graphical illustrations of the logical relationships as shown in

Figure 3.8:

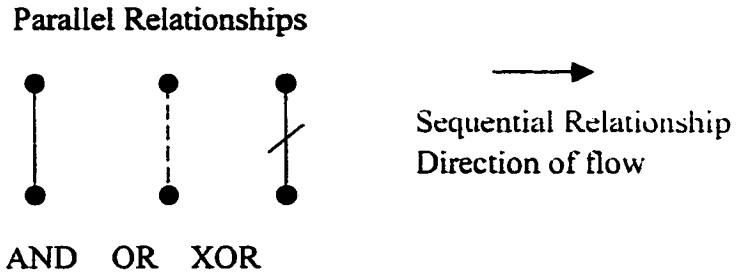


Figure 3.8. Graphical illustrations of logical relationships

Therefore, the assembly station in Figure 3.7 can be graphical illustrated in Figure 3.9.

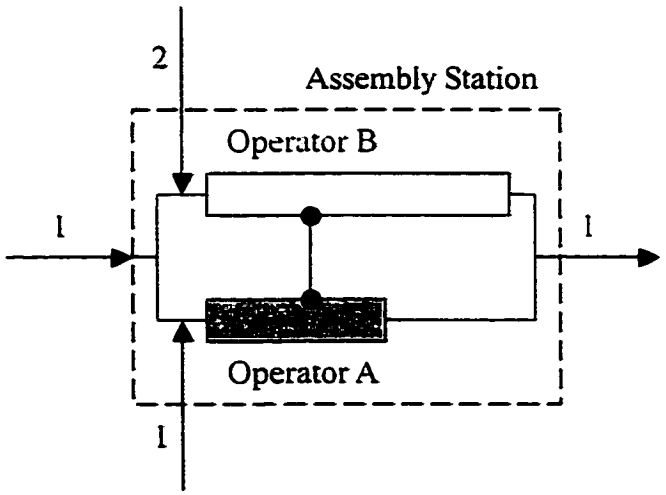


Figure 3.9. Graphical illustration of the assembly station in Figure 3.7

Operator A and B have different performance, which is represented by time to process, for their job.

The logical aspect of the model is therefore sound. However, a mechanism to implement the concept according to hierarchical modeling requirements is needed, e.g., they should be encapsulated and still provide uniform interfaces. This introduces the

Coupled Proto-Element (CPE), which implements coupling functions. Its structure is illustrated in Figure 3.10.

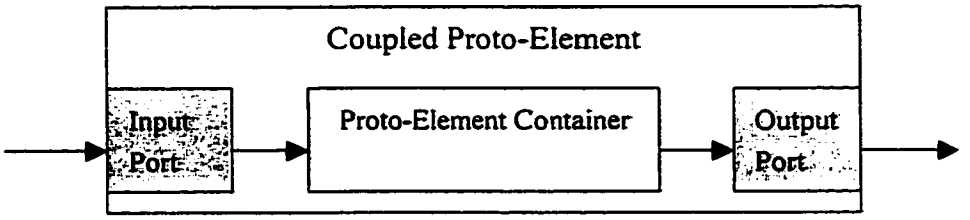


Figure 3.10. Coupled Proto-Element

The interfaces of the CPE, Input Port and Output Port, are implemented by PEs with zero processing time, this is because PEs have generic interfaces within the system, therefore it is easy to use them as interfaces to connect the PEs inside and outside the CPE. On the other hand, the pull and push actions are interfaces but do not have interfaces in themselves.

Therefore, the assembly station in Figure 3.9 can be contained within the PE Container and is seen as an ordinary PE from outside, as shown in Figure 3.11.

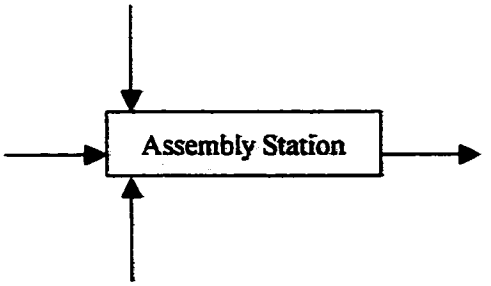


Figure 3.11. Assembly station is seen as a PE node from lower resolution

As the industrial and manufacturing systems are becoming more flexible, the control of such system has to depend on computers, in which control logics are expressed as rules. Rules have a basic form to express, which is "if....then....". For example, the FIFO and FILO logics can be expressed as

IF FIFO THEN put part at the end of queue

IF FILO THEN put part at the beginning of queue

Fortunately, the IF/THEN pair is a native function provided by any programming language. To model a conditional branching system, the rules can be

IF part.type == 2 THEN node4

IF part.type == 1 THEN node10

Therefore a machine with different performances for different part types can be modeled as:

Machine

IF part.type == 1 THEN { PE1; }

IF part.type == 2 THEN { PE2; }

If properly coded, users are able to change the rules during run time. However, this is a matter of how coding is done, which is out the scope of this thesis, so the author is not going to discuss it in detail.

3.2.3 Formal Representations of PE and CPE

The formal representation of PE and CPE is based on an extended Parallel DEVS (P-DEVS) formalism. The generic Parallel DEVS does not support the notion of logical relationships between internal operations. Therefore, the PE is going to be mapped into atomic model of P-DEVS in a straightforward manner; the CPE extends the definition of the coupled model of P-DEVS.

According to the P-DEVS, the PE is structured as

$$PE = \langle X_{PE}, Y_{PE}, S, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta \rangle \qquad (1)$$

where,

X_{PE} = {repair, pull_nextNode, push_foreNode,}

Y_{PE} = {push_pull}

S = {process, idle, working, failure, full, not_full, queue_size }

Variable = {capacity}

δ_{int} (queue_size = capacity) = full $\quad$ δ_{int} (queue_size < capacity) = not_full

δ_{int} (process, working) = idle

δ_{int} (working) = failure

δ_{ext} ((failure, e), repair) = working

δ_{ext} ((idle, working, not_full, e), push_foreNode) = (process, queue_size+1)

δ_{ext} ((working, (queue_size>0), e), pull_nextNode) = (not_full, queue_size-1)

δ_{conf} : always process δ_{int} first

λ (process, working) = push-pull

ta: time needed to process

The set of the states of the PE and their transitions are generated according to the states of real world resources. The *push* and *pull* actions are generated according to the discussion in previous sections. This formalism defines how the PE is going to respond to certain input. For example,

$$\delta_{ext}((\text{idle, working, not\_full, e}), \text{push\_foreNode}) = (\text{process, queue\_size+1}) \qquad (2)$$

means if the PE is working (not failure), idle and not full, when the preceding node pushes a workpiece into this node, this PE accepts it, increases its queue size by one and start processing it. When this PE finishes processing it, it will then follow the transition:

$$\lambda(\text{process, working}) = \text{push-pull} \qquad (3)$$

This will push the processed workpiece to the next node and pull another workpiece from preceding node. However, if in the next node, the transition (2) cannot be satisfied, the workpiece will stay in the current node.

There are two kinds of CPEs: tightly coupled PE (CPE(*t*)) and loosely coupled PE

(CPE(l)). The structure of CPE(l) is the same as the generic P-DEVS coupled model, which is illustrated in Figure 2.11. This form of CPE can be seen as a container of the independent PEs. The structure of the CPE(l) is as follows:

$$CPE(l) = \langle X, Y, D, \{PE_i\}, \{I_i\}, \{Z_{i,j}\} \rangle \quad (4)$$

where.

X is a set of input values

Y is a set of output values

D is a set of the DEVS component names

For each $i \in$ D,

PE_i is a DEVS component model

I_i is the set of influencees for I

For each $j \in I_i$

$Z_{i,j}$ is the i-to-j output translation function

Figure 3.12 illustrates the CPE(l).

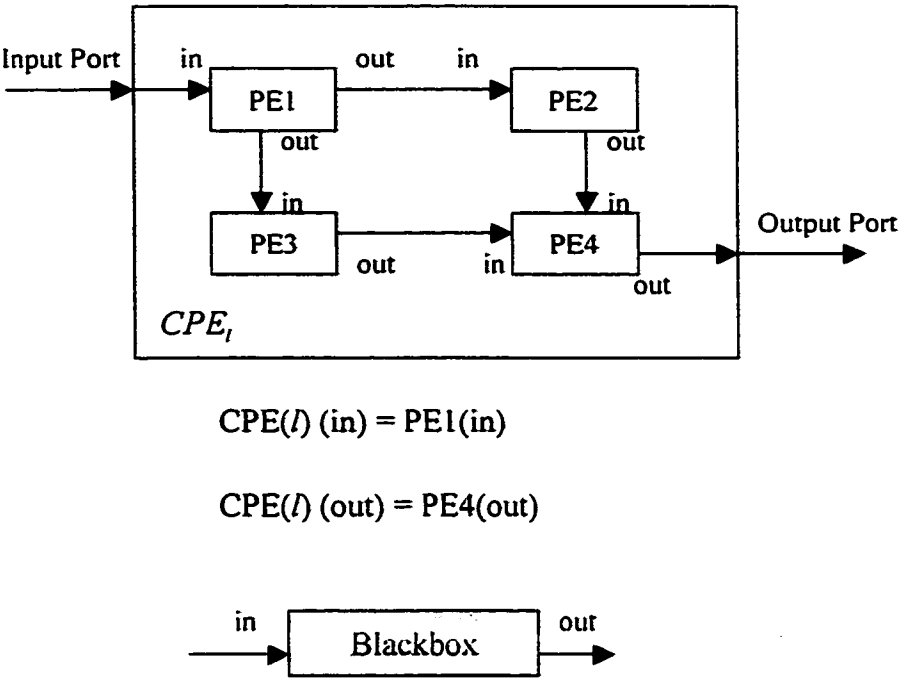


Figure 3.12. CPE(l) mapped to generic P-DEVS coupled model.

Therefore, the CPE(l) *is seen as a blackbox from system level.*

The CPE(t) is structured as follows:

$$CPE(t) = \langle X, Y, D, \{PE_i\}, \{I_i\}, \{Z_{i,j}\}, L \rangle \quad (5)$$

where,

X is a set of input values

Y is a set of output values

D is a set of the DEVS component names

For each $i \in$ D,

PE_i is a DEVS component model

I_i is the set of influencees for I

For each $j \in I_i$

$Z_{i,j}$ is the i-to-j output translation function

L = {ParallelAND, ParallelOR, ParallelXOR, Sequential}

The concern here is the time advance function and the output function of the coupled model that are related to the logical functions. This is developed as follows:

If L = ParallelAND, Then $tn_{CPE} = \max(tn_i)$

If L = ParallelOR, Then $tn_{CPE} = \min(tn_i)$

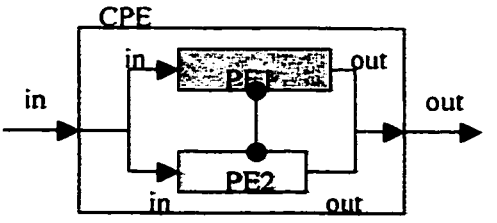
If L = ParallelXOR, Then $tn_{CPE} = \min(tn_i)$

If L = Sequential, Then $tn_{CPE} = \sum_{n=1}^{i} tn_n$

For every condition, $CPE_{out} = D_{tn_{CPE}}$

Therefore, the four CPE(t) is illustrated in Figure 3.13.

CPE(t) (ParallelAND)

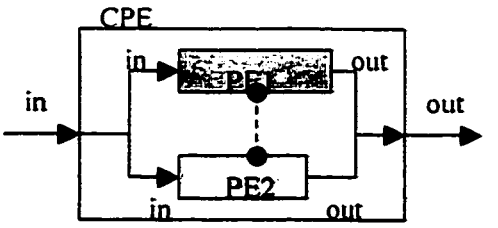


CPE(in) → PE1(in) CPE(in) → PE2(in)

If tn(PE1) >= tn(PE2) Then PE1(out) → CPE(out)

Else PE2(out) → CPE(out)

CPE(t) (ParallelOR)



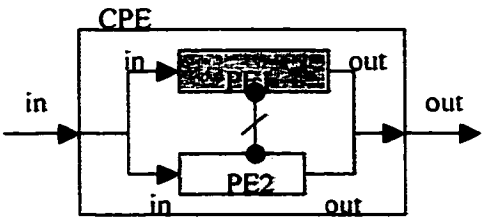
CPE(in) → PE1(in) CPE(in) → PE2(in)

If tn(PE1) >= tn(PE2) Then PE2(out) → CPE(out)

Else PE1(out) → CPE(out)

Figure 3.13. CPE(t) mapped into Coupled Model of Parallel DEVS

CPE(t) (ParallelXOR)



CPE(in) → PE1(in) CPE(in) → PE2(in)

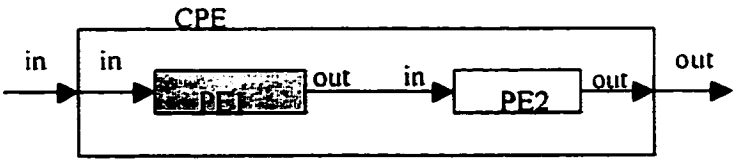
While tn(PE1) is not equal to tn(PE2) {

If tn(PE1) > tn(PE2) Then PE2(out) → CPE(out)

Else PE1(out) → CPE(out)

}

CPE(t) (Sequential)



CPE(in) → PE1(in) PE1(out) → PE2(in) PE2(out) → CPE (out)

Figure 3.13. (continued) CPE(t) mapped into Coupled Model of Parallel DEVS

3.2.4 Classes and Methods

In this section, the author derives a set of classes and methods based on the Object-Oriented approach. This classes and methods can be easily implemented in Java programming language. Outlines of the classes' definitions are given in pseudocode. Texts after the tags "//" are notes and explanations.

The Node class is developed as an abstract class to form the basic class structure of its subclasses. It is shown as follows:

```
ABSTRACT CLASS NODE
    Numerical Attributes:
        Capacity, Time to Process;
    Boolean Attributes:
        Idle, Busy;
```

The Proto-Element is developed as a subclass of Node and as the basic class for all other resources, its class structure is demonstrated in Figure 3.14.

```
CLASS PROTO-ELEMENT subclass of Node
    Numerical Attributes:
        Time to process = 0; //set default to take no time to process,
```

//same as a queue

Capacity = -1; // set default capacity to be unlimited

MTBF = -1; // set default to be impossible to fail

MTTR = -1; //because default MTBF is –1

Reject Rate = 0; //set default to be perfect

Boolean Attributes:

Idle = true; //initial status is machine available

Busy = false;

System Up = true; // initial system is running

System Down = false;

Full = false;

Empty = true;

FIFO = true; //default queue is first in first out

FILO = false;

All-At-A-Time = false;

Assemble = false;

List Attributes:

FromID, ToID; // to be continued

Methods: //continued from last page

Pull(), Push(),

Rules(), Entity_from(),

Schedule_next_event();

From();

Figure 3.14. Class structure of Proto-Element illustrated in pseudocode

Attributes are set with default

Therefore, the Proto-Element class, being the most generic class, have reserved attributes that are abstracted from all kinds of resources. Its subclasses – real-world resources can be modeled by *tailoring* Proto-Element – overriding necessary default values in the reserved attributes. The intention behind the "reserved attributes" is to construct uniform PE structure to increase the flexibility and ease the implementation. Therefore, each real-world resource is a specialized version of the PE, e.g., a queue has capacity and certain queueing rules; a conveyor has capacity and time needed to transport attributes; an AGV has capacity and time needed to transport attributes, but its All-At-A-Time value has to be set to true; a simple workstation can be modeled by capacity and time to process. From the structure point of view, the simple workstation and the conveyor are identical because their attributes are abstracted. However, from the users' point of view, it is necessary to distinguish them by giving them different names. This rises a motivation to develop the class hierarchies in two layers, one is implementation layer, which distinguishes classes by their features; the other layer is user interface layer, which distinguish classes by their names in real world. This is illustrated in Figure 3.15. The Coupled Proto-Element is constructed in the implementation layer. However, due to its

functionality, it can also be seen from user-interface layer.

Result of this two-layer construction is that it generates a framework for pushing down much of the tedious "low-level" aspects of simulation to the systems level, and away from the user.

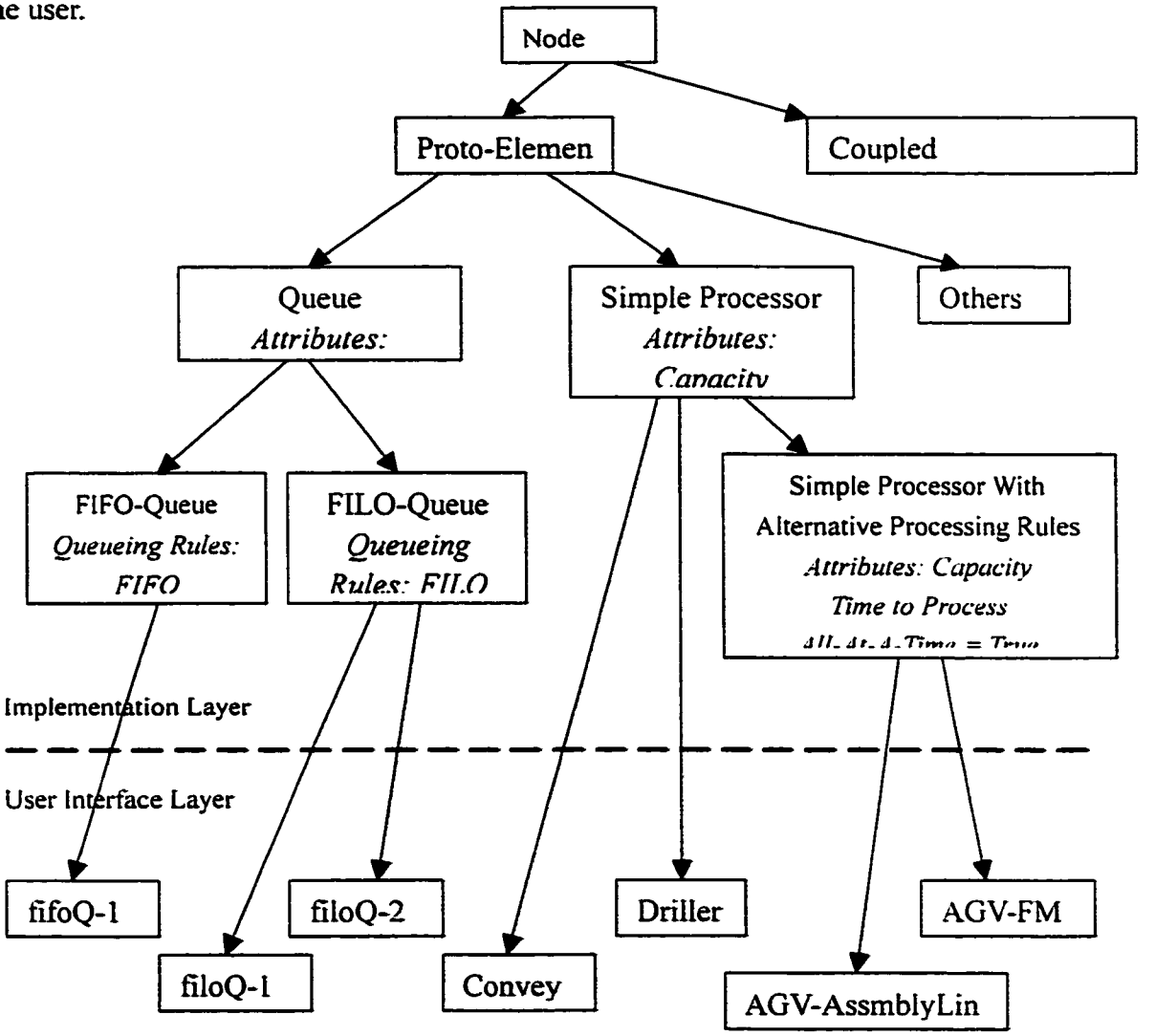


Figure 3.15. Two layers of Class hierarchies

The structure of the Coupled PE (CPE) is illustrated in Figure 3.16. According to the formalism of the DEVS, the CPE shares the same attributes and behaviour as the PE. Its

behaviour is implemented by the methods in Java. The difference is that CPE does not have attributes that are related to queue, and it adds necessary methods to implement the coupling mechanism.

CLASS COUPLED PROTO-ELEMENT subclass of Node

Numerical Attributes:

```
Time to process = 0;  //set to take no time to process, same as a
                      //queue
Capacity = -1;        // set default capacity to be unlimited
MTBF = -1;            // set default to be impossible to fail
MTTR = -1;            //because default MTBF is –1
Reject Rate = 0;      //set default to be perfect
```

Boolean Attributes:

```
tc = true;            //set default to tightly coupled
Idle = true;          //initial status is machine available
Busy = false;
System Up = true;     // initial system is running
System Down = false;
Full = false;
Empty = true;
Assemble = false;
```

List Attributes:

FromID, ToID;

Input_port, Output_port; //implemented by PEs

Methods:

Rules(), Entity_from();

From();

Methods For Logical Relationships:

SequentialPEs:

SequentialPE (PE1, PE2);

ParallelAndPEs:

ParallelAndPEs (PE1, PE2);

ParallelOrPEs:

ParallelOrPEs (PE1, PE2);

ParallelXorPEs:

ParallelXorPEs (PE1, PE2);

Figure 3.16. Class structure of the Coupled PE.

There are two unique nodes that have to be treated separately, Entity Generator and Entity Destructor. The Generator is the node in charge of generating the entities to feed the system; the Destructor is the one in charge of collecting entities that leave system and destroying them after that. Different from other nodes, the Generator only pushes the entities to its following nodes and never does the pull action, while the Destructor only has

a destruct function and never does pull and push action. Hence their event descriptions have different structures. Figure 3.17 describes the actions of the Generator:

Generator Event Description:

a. Generate an entity

b. Push the entity to next node;

c. Scheduling next event to generate entity

Destructor Event Description:

a. Accept an entity/entities in to current node.

b. Destroy entity/entities

Figure 3.17. Event description of the Generator and Destructor

Therefore, the Generator is outlined as follows:

CLASS GENERATOR subclass Node

Numerical Attributes:

Amount to generate

Time limit

Interarrival time

Part type to generate

List Attribute

ToNodes

Methods:

Generate Entity

Push to Next Node

Schedule Next Event

A set of classes is generated to support the simulation engine and exist as the basic components of the simulation system. For example, the random number generator and the statistics class.

From program structure point of view, it is essential that any simulation system should have the ability to provide statistical distributions. So a *Distribution* class is needed. This class is in charge of generating random numbers by the statistical distributions required by the user. It should have all the distributions, e.g., Uniform, Erlang, Exponential, Poisson, etc.

From the functionality point of view, a simulation system should be able to give statistical reports for users to analyze the simulated system. Therefore a *Statistics* class is needed to gather statistical information and to generate reports.

Figure 3.18 illustrates the hierarchy of the classes in the implementation layer.

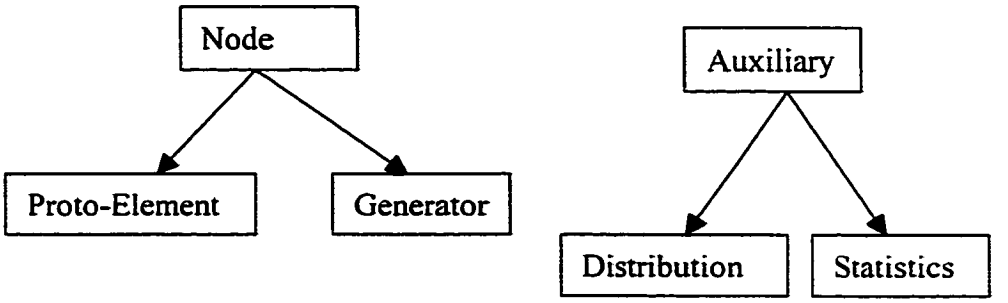


Figure 3.18. Class hierarchies in Implementation Layer

3.2.5 Possibilities of Distributed Computing

Since the author's work is within the modeling methodology, to find out the best distributed computing algorithm or to develop a distributed computing mechanism is out of the scope of this thesis. However, it is worthwhile to prove that it is possible to utilize the power of distributed computing in DOBIS. This is the intention of this section.

The classes defined in the last section are purely for modeling the manufacturing systems. In this section, the author presents a possible framework that is capable of implementing the distributed computing to speed up the simulation execution and provides modular structure so that it is possible to adopt advanced distribution algorithm in the future.

Suppose the framework is in Server/Client mode. The server is the mechanism that holds the model information (Figure 3.19), the client is a computing unit that accepts computing tasks and communicates with the server. There should be several modules on server, which are Modeller, Partitioner, Task Distributor and Logical Process. The Logical

Process implements the distributed computing algorithm, which is the Time Warp algorithm selected by the author for illustration. The Modeller is a modeling environment that implements the system-level and object-level modeling based on the Node-Based strategy, the PE and the classes that the author has introduced in the section 3.2.1 and section 3.2.2. The Partitioner is in charge of partitioning the model into small Logical Processes (LP). Then the Task Distributor distributes the LPs to the resources over the network. The Logical Process module takes care of all the communications and guarantees the causality among the processes, it also drives the simulation to the end.

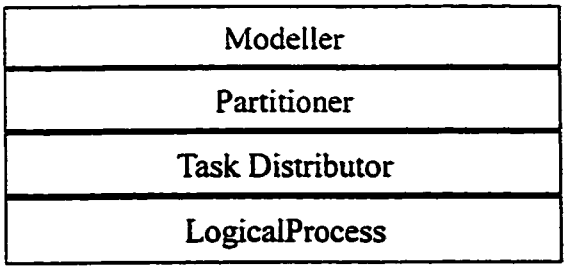


Figure 3.19. The Server structure

The distributed algorithm selected has direct impact on the partitioning strategy. Since this proposed system performs distributed computing at the component level, each of the resources should reside on one computer to perform the simulation. But sometimes it is not practical to make such arrangement. The author's solution is to use the thread mechanism, which is provided by the Java programming language. A thread is a part of a program that can execute independently of other parts. An operating system that has the ability to execute more than one task is called multi-task operating system. The multi-task system

separates the CPU time into many slices, and each of the slices takes care of a thread. It enables programmers to design programs whose threaded parts can execute concurrently. In today's computing environment, multi-task operating systems are everywhere, e.g., Unix, Linux, Windows9X, WindowsNT, MacOS and OS/2. In the context of this proposal, one thread implements one LP. An LP is a process that resides on a thread, which takes a small portion of the model and keeps the causality between this LP and others according to the model information. Unfortunately there is no general partitioning strategy available and most of the partitioning is done manually, which means the partitioning is done before the coding.

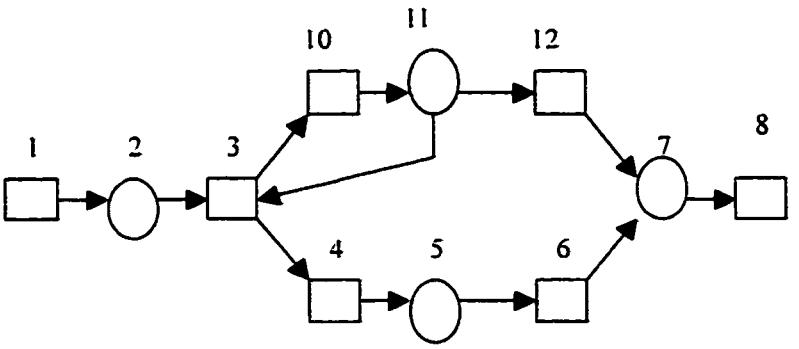


Figure 3.20. An example of manufacturing line

Since the model that is generated by the proposed simulation system consists of objects with uniform structures, it is possible to partition the model into small LPs by going through general and simple rules. Consider a material flow layout as shown in Figure 3.20. suppose every workstation has a queue before it. The queues do not change the physical property of the parts, only the workstations do. If a queue receives a message with

timestamp t lower than its LVT (local virtual time), there will be two possible situations. One situation is that t is lower than the timestamp of the part that is being processed by the workstation. In this case both the queue and the workstation have to be rolled back. However, if t is lower than the LVT but higher than the timestamp of the first part in the queue, the rollback will be as easy as changing the sequence of the queue to insert the new part. Therefore, if we put a queue before a workstation, the rollback overhead can be reduced. This mechanism is illustrated in Figure 3.20.

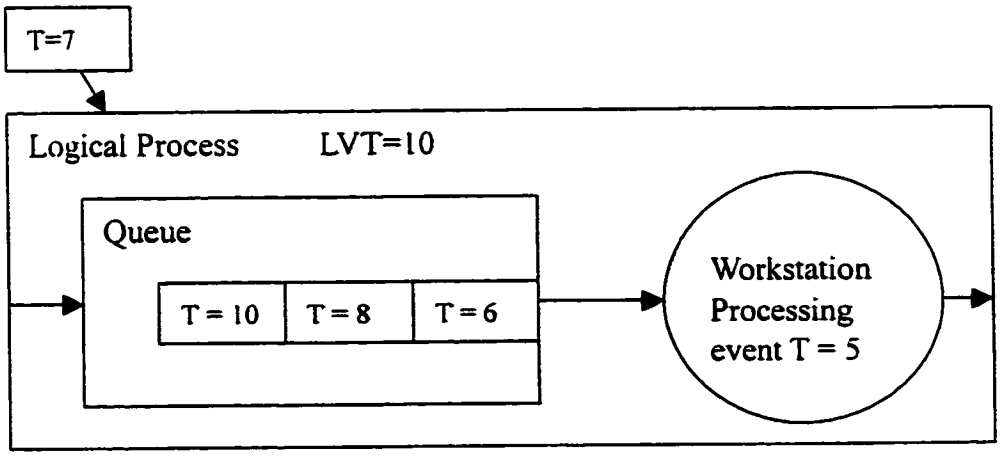


Figure 3.20. An illustration of the partitioning strategy. T is the timestamp of the part.

Therefore, a workstation node and its immediate preceding queue node or nodes reside in the same LP, and therefore the same thread. A queue node with branching function, for example the node 3 in Figure 3.20, is likely to generate more complex causal relationships with its following nodes and have more extensive influences once a rollback is executed, and therefore generates more rollback overhead. Hence the LP of the branching queue node

should be extended until the first workstation nodes of each of its branches, see illustration in Figure 3.21. Since this approach follows straightforward logics, it is possible for the computers to do the partitioning tasks automatically.

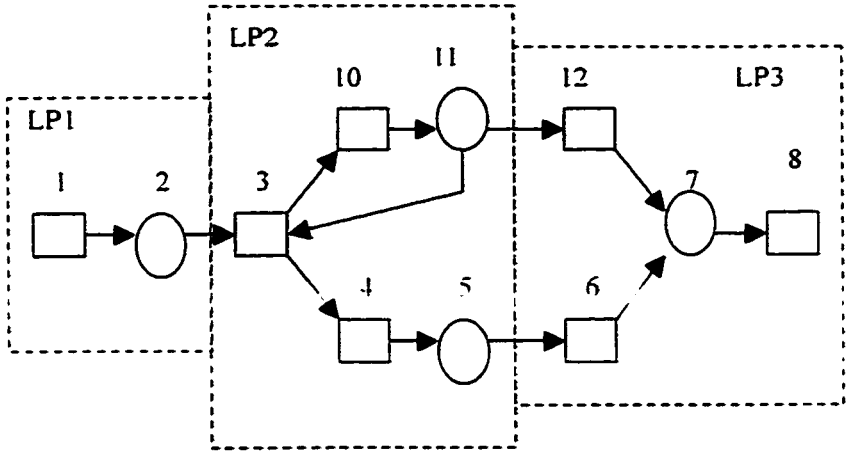


Figure 3.21. An illustration of the model partitioning. The dash line rectangles are logical processes

A LogicalProcess class should be generated as an executor to perform the simulation. It needs several methods and data structures to implement the distributed computing mechanism. InputChannel and OutputChannel implement the communication with other LPs: InputQueue. OutputQueue and StateRecord implement the TimeWarp algorithm. The proposed class structure and hierarchy is shown in Figure 3.22. The client side has the LogicalProcess module in order to work properly with the server.

CLASS LOGICAL-PROCESS

Numerical Attributes:

Local Virtual Time;

Global Virtual Time;

Incoming Part Timestamp;

Boolean Attributes:

RollBack = false; // the default value

Rearrange Sequence = false; // the default value

List Attributes:

InputMessageQueue;

OutputMessageQueue;

StateRecordQueue;

Methods:

InputChannel;

OutputChannel;

Figure 3.22. The structure of the class Logical Process

3.3 A Comparison Of DOBIS and Existing Simulation Systems

In this section, a comparison of DOBIS and the existing commercial simulation systems is given in a table (Table 3.1). This comparison emphasizes on the modeling aspects of the systems, and is extended to ease-of-use and computing aspect. The applications of the non-general purpose systems are also listed.

Table 3.1. A comparison of DOBIS and existing systems

	System	DOBIS	AutoMod [37]	GPSS [55,62]	Silk[78]	Arena/SIMAN [29]
Functional	OO Paradigm*	Y	N	N	Y	N
	Modeling Strategy	ES	PI	PI	PI	PI
	Hierarchical Modeling	Y	N	N	N	N
	Distributed Computing	Y**	N	N	N	N
Consistency with mental models	Consistent Syntax For All Kinds of Objects	Y	N	N	N	N
	Open Structure for Object/System	Y	N	N	N	N
Usability, Flexibility and Extensibility	User Can Access the Structure of the Object	Y	Y***	Y***	Y	Y***
	Modification to Structure Requires Low Level Coding	N	Y	Y	Y	Y
	Reusability Achieved in Structure Level****	Y	N	N	N	N
	Platform Independent	Y	N	N	Y	N

EI: Event-Scheduling Strategy; PI: Process-Interaction Strategy

Functional performance is the things it can do; usability is the ease with which it can be instructed to do things; consistency with mental models means it lets people model systems however they picture the system rather than teaching them to think in a certain way.

** Apply Object-Oriented paradigm in modeling, not in the construction of the system. Whether a simulation language is Object-Oriented can be seen from how it creates and describe new objects [29].*

*** the flexibility to support distributed computing has been embedded in the framework, but that it still has to be implemented.*

****this functionality is achieved through sub-routine and is not provided directly in the system.*

***** DOBIS reusability is achieved in structure level, whereas existing OO simulation system achieves in object level. See example 4 for further detail.*

The advantages of DOBIS to the users should be distinguished from that to the developers of the overall Intelligent Simulation project. To the developers, the advantages of DOBIS lie in that the coding of the system is inherently controllable.

In order to compare the working procedure of the systems, a single-server example from [37] is given below.

EXAMPLE 1: In this hypothetical facility, loads are created with an inter-arrival time that is exponentially distributed with a mean of 12 minutes. Loads first go to a sorting process, where each load is sorted for a time that is uniformly distributed between 8 and 12 minutes. The sorted loads then go to a processing station that processes loads in three steps for the amount of time shown in the table below:

Step	Time
Preparation	Constant 100 seconds
Processing	Uniformly distributed between 3 and 13 minutes
Unloading	Constant 20 seconds

After the unloading step, loads leave the system.

Figure 3.23 illustrates procedure of the processes in the example.

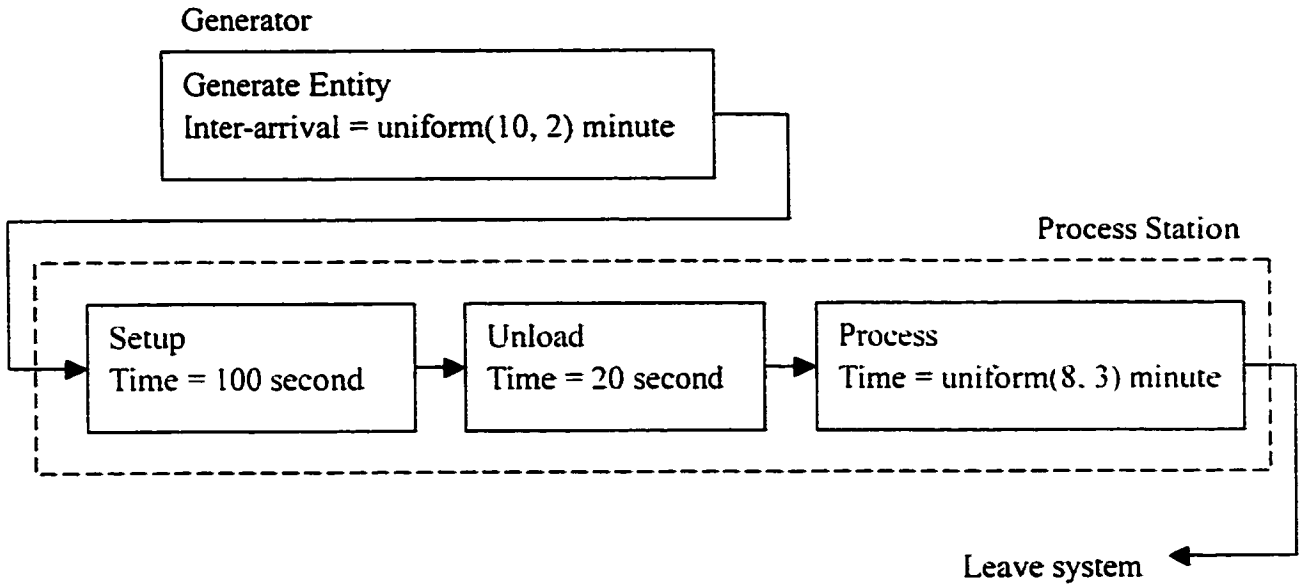


Figure 3.23. Processes in example 1.

The AutoMod model of this example is shown in Figure 3.24.

```
begin P_sort arriving procedure      /* All procedures start with begin */
  wait for uniform 10, 2 min         /* Delay load for 8 to 12 minutes */
  send to P_procstation              /* Sends load to another process */
end                                  /* All procedures end with an end */

begin P_procstation arriving         /* "procedure" syntax is optional */
  wait for 100 sec                   /* Time units (seconds) are specified */
  wait for uniform 8, 5 min          /* Delay load for 3 to 13 minutes */
  wait for 20                        /* Default time units are seconds */
  send to die                        /* The load leaves the simulation */
end
```

Figure 3.24. AutoMod model of example 1.

```
GENERATE     10,2
SEIZE        ProcessStation
ADVANCE      1.67
ADVANCE      8,5
ADVANCE      0.33
RELEASE      ProcessStation
TERMINATE    1
```

Figure 3.25. GPSS model of example 1.

The GPSS model is illustrated in Figure 3.25.

The model generated by DOBIS is shown in Figure 3.26.

```
New PE:  Setup, Process, Unload;              // declare three PEs
  Setup.time_to_process = 1.67;                  //setup time is 100 seconds
  Process. time_to_process = Distribution.uniform(8, 5);
  //processing time is uniform distributed between 3 and 13 miniutes
  Unload. time_to_process = 0.33;  //unloading time is 40 seconds
New CPE: ProcessStation;                        //declare one CPE
  ProcessStation.Sequential(Setup, Process, Unload);
  //construct the logical relationships of the PEs in the CPE
New Generator: driver;                          //declare the entity generator
  driver.InterArrival = Distribution.uniform(10, 2);    //inter-arrival time
////////////////////////////////////////////////////////////////////////////////end of object definition
ProcessStation.fromID = driver;                 //define the routing information
Start;
```

Figure 3.26. Model of example 1 generated by DOBIS.

(*in pseudocode*)

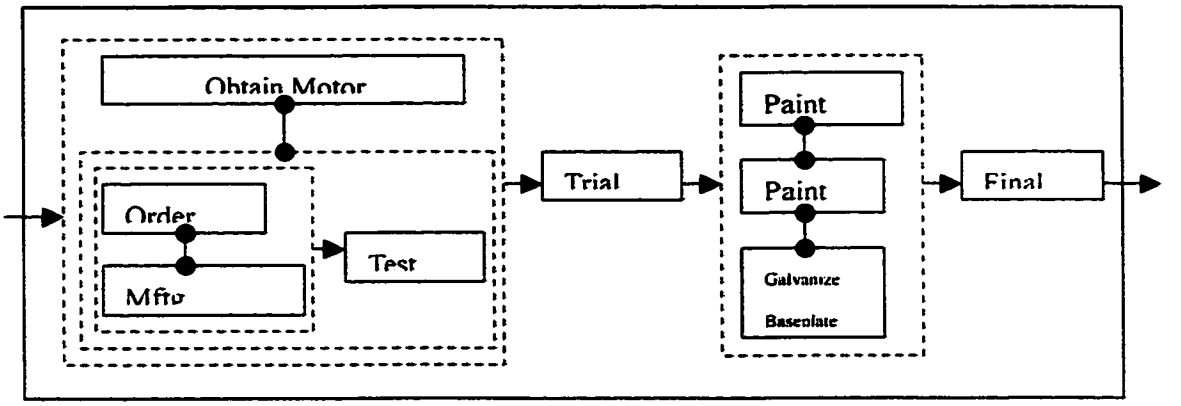
The example 1 represents a rather simple scenario, where there is no parallel activities, and only one stream of flow entities exists in the system at any time. We can see that all the three systems are able to provide solutions. However, there are some differences in their approaches. AutoMod model has clear boundaries for each object. e.g., processes are

separated by the *begin* command for each station; GPSS model is compact; the model generated by DOBIS is the only one explicitly defines the logical relationships between internal operations, and implements the Object-Oriented paradigm.

A significant difference can be seen from modeling a complex scenario, e.g., which has parallel activities involved. Let us study another example from [83]:

EXAMPLE 2: A manufacturer makes centrifugal pump units which are assembled to customer orders. The orders arrive on average, every 5 hours, exponentially distributed. When the order arrives, two copies are made. The original order is used to obtain a motor from stock and prepare it for assembly (200±100 minutes). The first copy is used to order and adapt a pump (180±120 minutes), the second copy is used to initiate the manufacture of the baseplate (80±20 minutes). When the pump and the baseplate are ready, a test fitting is carried out (50±10 minutes). All three components are assembled (60 minutes), when they are available. The unit is then dismantled, and the pump and motor are painted (100±20 minutes and 120±30 minutes respectively), and the baseplate is galvanized (120±30 minutes). Final assembly then takes place (150±30 minutes).

This whole process can be illustrated as following graph:



Therefore, the model generated by DOBIS is:

New PE: Obtain_Motor, Order_Pump, Mftg_Baseplate, Test_Fitting, Trial_Assm, Paint_Motor, Paint_Pump, Galvanize_Baseplate, Final_Assm;

Obtain_Motor. time_to_process = Distribution.uniform(200, 100);

Order_Pump. time_to_process = Distribution.uniform(180, 120);

Mftg_Baseplate. time_to_process = Distribution.uniform(80, 20);

Test_Fitting. time_to_process = Distribution.uniform(50, 10);

Trial_Assm. time_to_process = 60;

Paint_Motor. time_to_process = Distribution.uniform(100, 20);

Paint_Pump. time_to_process = Distribution.uniform(120, 30);

Galvanize_Baseplate. time_to_process = Distribution.uniform(120, 30);

Final_Assm. time_to_process = Distribution.uniform(150, 30);

New CPE: Order_Mftg, Test_Fit, Motor_Assm, Paint_ Galvanize, Pump_Assm;

Order_Mftg.ParallelAND(Order_Pump, Mftg_Baseplate);

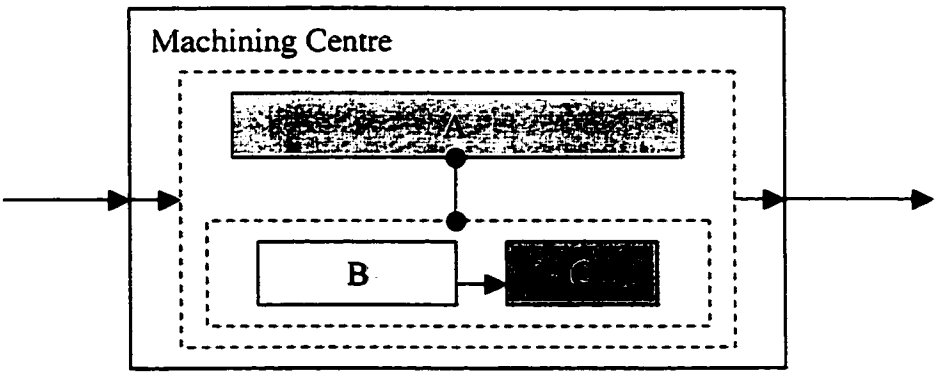
Test_Fit.Sequential(Order_Mftg, Test_Fitting);

```
Motor_Assm.ParallelAND(Test_Fit, Obtain_Motor);
Trial_Assm.from(Motor_Assm.Output_port);
Paint_ Galvanize.ParallelAND(Paint_Motor, Paint_Pump, Galvanize_Baseplate);
Paint_ Galvanize.Input_port.from(Trial_Assm);
Final_Assm.from(Paint_ Galvanize.Output_port);
Motor_Assm.Input_prot.from(Pump_Assm.Input_port);
Pump_Assm.Output_port.from(Final_Assm);
```

The GPSS code for simulating this scenario is shown in Appendix A. The steps to go through in GPSS is more than twice of DOBIS's. This indicates that modeling parallel activities in GPSS is more complicated. This conclusion is proved in the discussion about modeling parallel activities in [84]. One may arguer that it really depends on what those lines of code cause to happen computationally. However, since at this time, the system is designed for research purposes, then the author can assume there will be people developing software that uses DOBIS who will simply be typing text into files. Short programs are easier to read and understand than long ones. So lowering the number of lines of code can be seen as a possible means to improve readability of the code. Furthermore, one may assume that a graphical user interface for a DOBIS based system would 'translate' graphical symbols into text ones (in Java, say) which would then be compiled. By keeping the text language simple, one may expect that the GUI will be easier to develop.

Other than its ability to describe parallel activities, DOBIS has unique flexibility to meet the changing world. See the following examples:

EXAMPLE 3: A machining centre of a large-scale production line has three major components, A, B and C. All the components have to be working in order for the machining centre to work. The logical relationships between A, B and C is shown below:



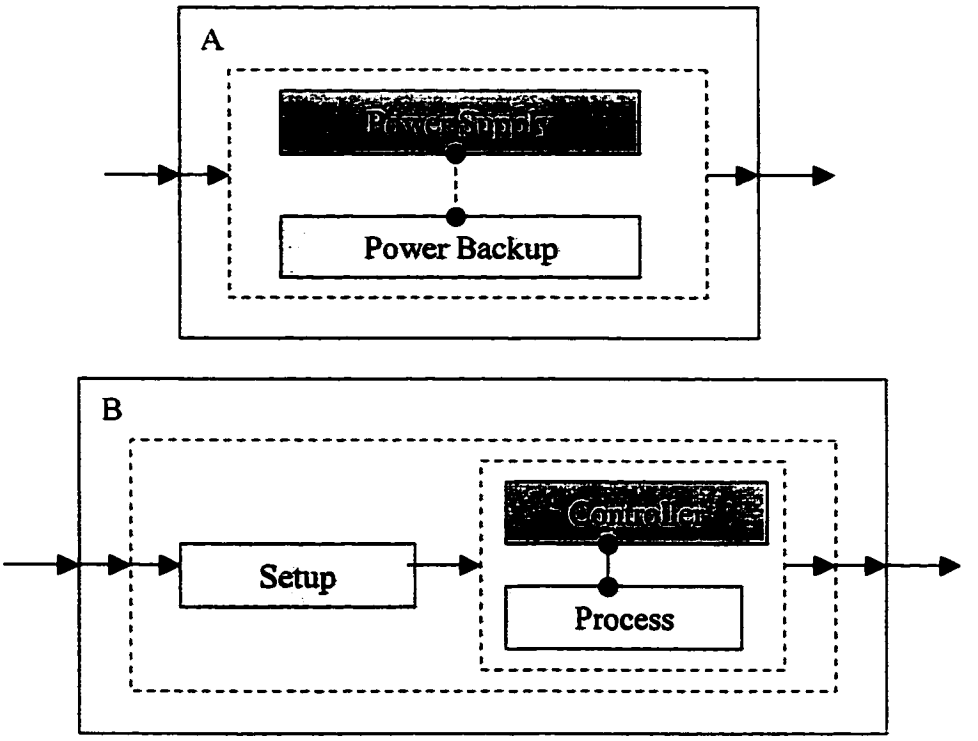
B and C are the components actually process the workpieces. Their processing times as well as the MTTFs and MTBFs are shown in the following table:

Component	Processing Time	MTBF	MTTR
A	N/A	15000	30
B	Constant(10)	8000	20
C	Uniform(10, 3)	6000	10

Therefore, the model generated by DOBIS is:

```
New PE: A, B, C
  A.MTBF = 15000;
  A.MTTR = 30;
  B.MTBF = 8000;
  B.MTTR = 20;
  C.MTBF = 6000;
  C.MTTR = 10;
  B.Time_To_Process = 10;
  C.Time_To_Process = Distribution.uniform(10, 3);
New CPE: Machining_Centre, B_C;
  B_C.Sequential(B, C);
  Machining_Centre.ParallelAND(A, B_C);
```

EXAMPLE 3 Continued: After studying for some time, the user wants to get to higher levels of details of A and B. The B is decomposed into 3 internal operations, Setup, Process and Controller; component A is modified and added with a backup equipment and therefore is decomposed into 2 internal operations. Their processes relationships are shown below:



Ignoring the trivial numerical attributes assignments, the structure of the machining centre is modeled as following:

New PE: C, Power_Supply, Power_Backup, Controller, Process, Setup;

New CPE: A, B, Machining_Centre, Controller _ Process, B_C;

A.ParallelOR(Power_Supply, Power_Backup);

Controller _ Process.ParallelAND(Controller, Process);

B.Sequential(Setup, Controller _ Process);

B_C.Sequential(B, C);

Machining_Centre.ParallelAND(A, B_C);

This is a "top down" approach. With the recursive decomposition ability provided by hierarchical modeling, users are able to model any system with arbitrary level of detail. In addition, the Proto-Element provides uniform structure to capture any internal operations,

simplifies the abstraction of the real-world objects.

In contrast, implementing this application in AutoMod and GPSS are not as easy as DOBIS is. Because of their native sequential model description characters, considerable effort can be required to accomplish the goal, e.g., new model descriptions may require modifications made to existing codes.

EXAMPLE 4: According to the author's proposal, systems, resources and internal operations are different views of the same thing. Therefore, systems at different levels may share the same structures. For example, an assembly line and a machining centre illustrated in Figure 3.27.

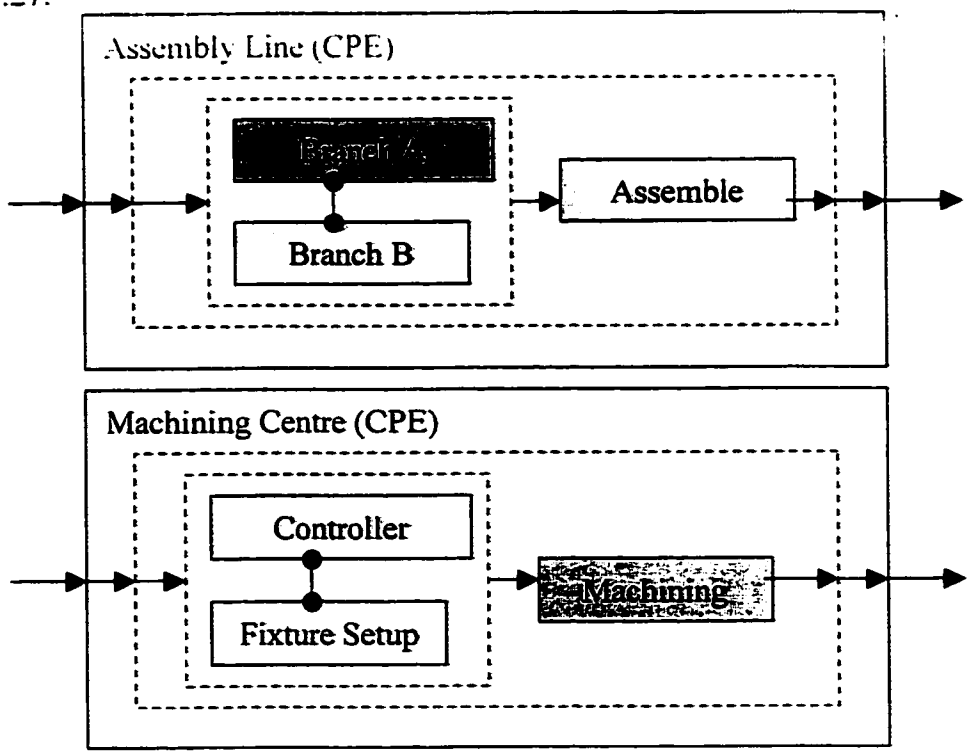


Figure 3.27. Structures of an assembly line and a machining centre

These two CPEs are different in user interface layer (Figure 3.15), but they are same in the implementation layer. Therefore, once a CPE for the machining centre is constructed, it can be used as the assembly line. This is a feature that none of the existing systems has achieved.

It is worthwhile to revisit the multiple inheritance versus single inheritance at this stage.

3.4 Multiple Inheritance v.s. Single Inheritance Revisited

In traditional simulation systems, objects are distinguished from each other. Therefore, many objects that are "naturally mapped" from real world have distinct structures and functions. Furthermore, these structures are fixed when they are constructed. Therefore, the only way to generate new objects and reuse existing codes is through *inheritance* provided by the Object-Oriented Programming Language. In many cases, single inheritance is not enough for such a modeling environment because the new objects have common characters across several different kinds of existing objects (Figure 3.28), this is the reason why multiple inheritance is important to existing simulation systems.

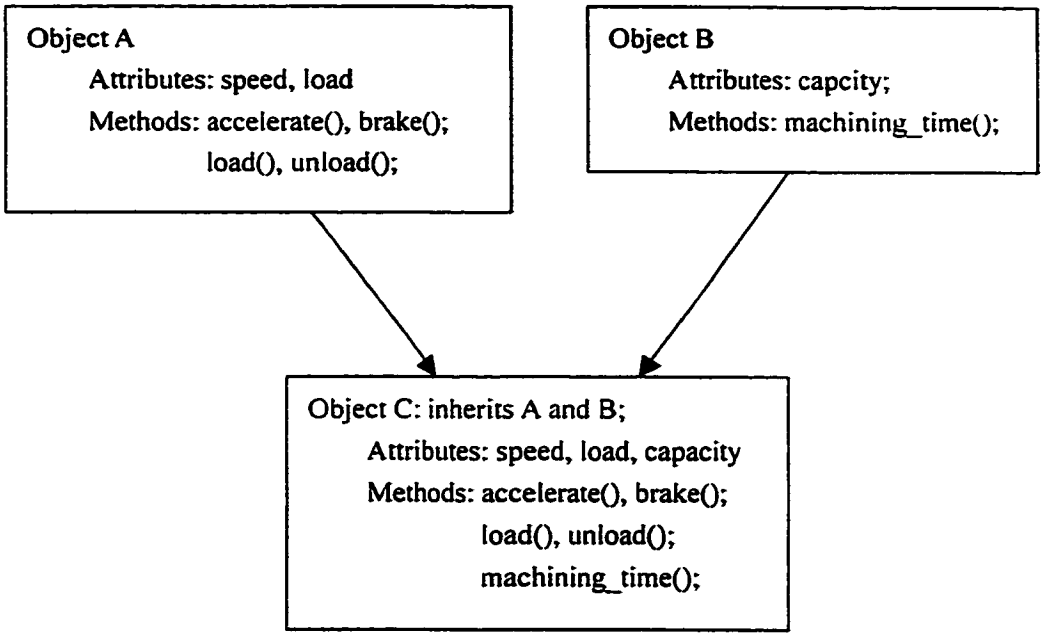


Figure 3.28. Multiple Inheritance

In DOBIS, because systems and resources are recursively decomposed, objects in the system have flexible structures. New objects are not constructed from inheritance; instead they are *compositions* of existing structures. This is because they are all constructed by identical PEs, and the PEs are the most basic element used to model any process. In other words, DOBIS reusability is achieved in *structure* level, whereas existing OO simulation system achieves in *object* level. A graphical illustration provides better picture of this concept, see Figure 3.29.

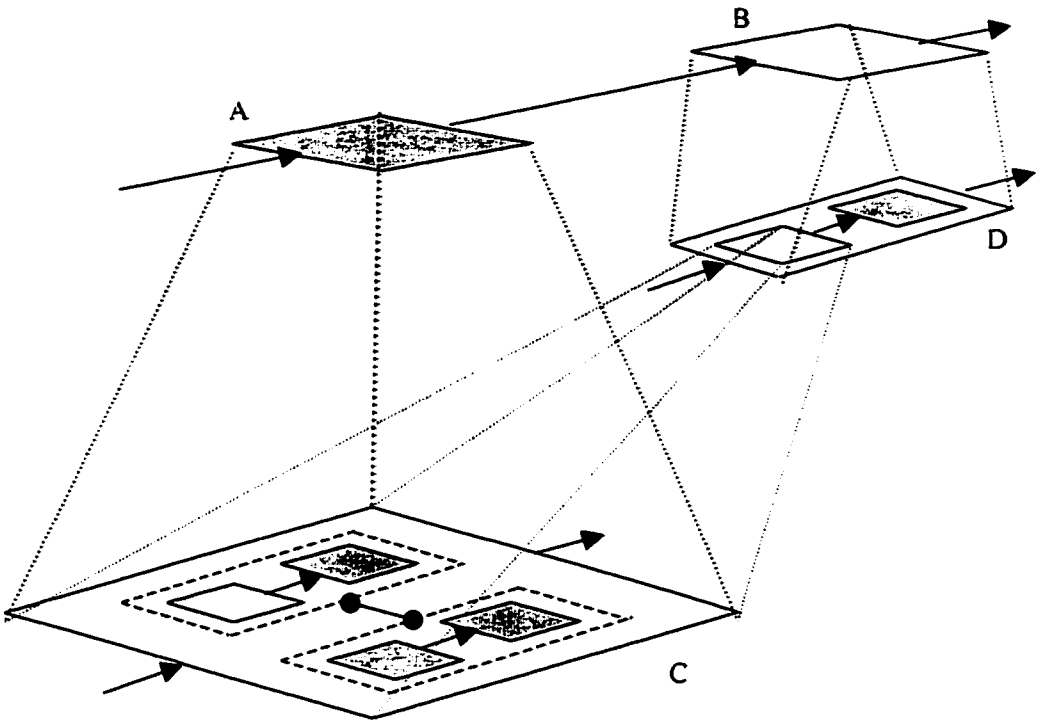


Figure 3.29. Object compositions in DOBIS

In this graph, C constructs A and part of D. The structure of C can be reused at different levels of detail.

From another aspect, in DOBIS, the constructions of new objects are implementations of the modeling methodology, whereas other systems depend on the Object-Oriented paradigm.

However, DOBIS still supports constructing new objects through single inheritance when it is necessary. Therefore, if we view constructing new objects through inheritance as two-dimensional (Figure 3.28), the construction of new objects in DOBIS is three-dimensional (Figure 3.29).

CHAPTER FOUR

AN EXAMPLE IMPLEMENTING THE NEW FRAMEWORK

In this chapter, the author presents an example to demonstrate the flexible modeling ability of the proposed framework.

Figure 4.1 shows a portion of a real water heater production line in SunPu Electrical Appliances Company, Beijing, China. The layout shows the workstations only.

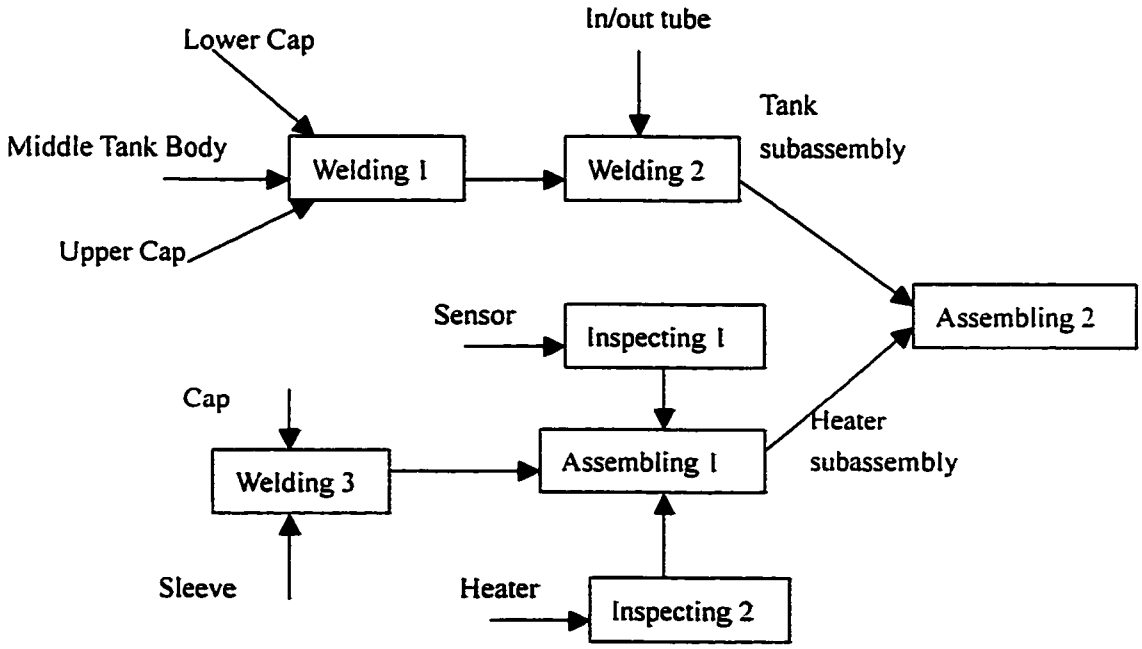


Figure 4.1. A water heater production line in SunPu, Beijing, China

To translate this system into Node-Based strategy and partition them into Logical Processes by graph, see Figure 4.2.

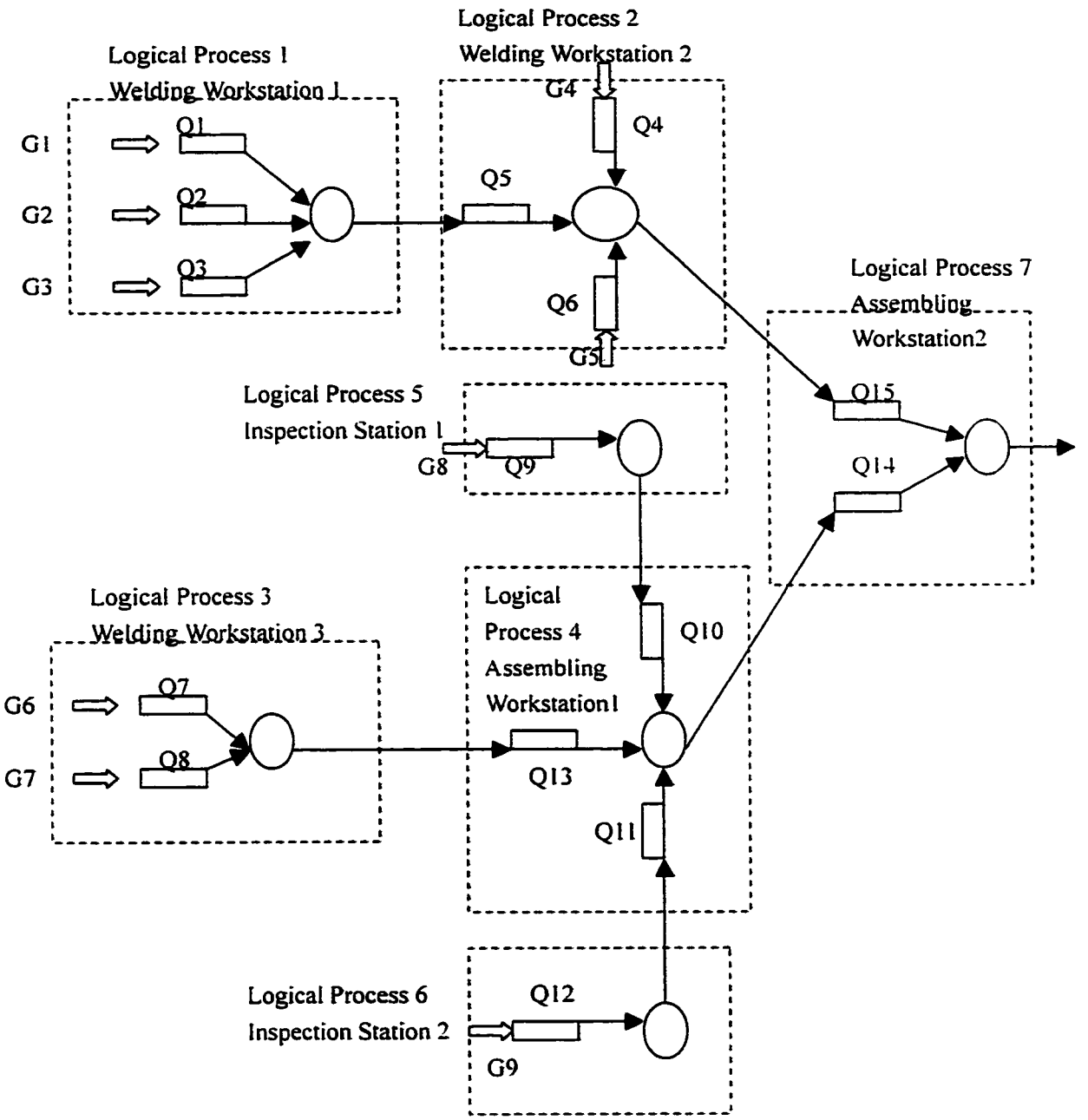


Figure 4.2 Graph shown in Node-Based strategy and partitioning of Logical Process.

Dash line rectangles illustrate the partitioned LPs.

As shown in the layout, there are totally nine kinds of different entities interacting each other as well as the resources. The shaded wide arrows in Figure 4.2 illustrate the entity

generators.

At welding workstation 1, three different parts are needed. Two welding robots A and B do the welding. One human operator is in charge of inspecting the parts before welding. Once the human operator finishes the inspection, the two robots will then do the welding and finally pass the part into next queue. Because the management wants to know the detail performance of each of the robots and the operator, it is desired to model the workstation in detail. Suppose the time for the human operator to finish his job is *time1*, the time for robot A and B to finish their job are *time2* and *time3* respectively, MTBFs for robot A and B are *time4* and *time5* respectively. Both robots are needed to do the job. This workstation can be modeled as following:

```
New PE: Inspector, RobotA, RobotB;
New CPE: WeldingStation1,Robots;
    Inspector.ProcessingTime = time1;
    Inspector.Capactiy = 1;
    RobotA.MTTF = time4;
    RobotA.ProcessingTime = time2;
    RobotA.Capactiy = 1;
    RobotB.MTTF = time5;
    RobotB.ProcessingTime = time3;
    RobotB.Capactiy = 1;
```

Robots.Assemble = true;

Robots.ParallelAND(RobotA, RobotB);

WeldingStation1.Sequential(Inspector, Robots);

In welding station 2, things are slightly different: no human operators, but two welding robots with different setup times. Once both of the setups are finished, the two robots perform the welding operation simultaneously and have the same distribution of time to finish the work. Robot C setup time is *time6*, robot D setup time is *time7*. Operating time for both of them is *time8*. It can be modeled as following:

New PE: SetupC, SetupD, ProcessC, ProcessD;

New CPE: RobotC, RobotD, WeldingStation2;

SetupC. ProcessingTime = time6;

SetupD. ProcessingTime = time7;

ProcessC.ProcessingTime = time8;

ProcessD.ProcessingTime = time9;

RobotC.Sequential(SetupC, ProcessC);

RobotD.Sequential(SetupD, ProcessD);

WeldingStation2.Capactiy = 1;

WeldingStation2.Assemble = true;

WeldingStations2.ParallelAND(RobotC, RobotD);

The welding station 3 is simpler than the other two stations. Two parts come in and are welded together by one robot. Setup time is *time9* and operating time is *time10*, MTTF is *time11*.

New PE: SetupWeldingStation3, ProcessWeldingStation3;

New CPE: WeldingStation3;

WeldingStation3.MTTF = time11;

SetupWeldingStation3. ProcessingTime = time9; //time to setup

ProcessWeldingStation3.ProcessingTime = time10; //time to operate

WeldingStation3.Capactiy = 1;

WeldingStation3.Assemble = true;

WeldingStation3.Sequential(SetupWeldingStation3, ProcessWeldingStation3);

The models of inspection stations are simpler compared to the welding station 1 and 2. The total processing times for the two inspection stations are *time12* and *time13*. Two inspection stations have the same characters. The model is as following:

New PE: InspectionStation1, InspectionStation2;

InspectionStation1.ProcessingTime = time12;

InspectionStation2.ProcessingTime = time13;

The assembly workstation 1 and assembly workstation 2 do their works in a sequential manner, so there is no "Parallel" in the model. Human operators do the assembling works.

one on each of the stations. Operating time is time14 and time15 respectively.

New PE: AssemblyWorkstation1;

AssemblyWorkstation1.ProcessingTime = time14;

AssemblyWorkstation1.Capactiy = 1;

AssemblyWorkstation1.Assmeble = true;

New PE: AssemblyWorkstation2;

AssemblyWorkstation2.ProcessingTime = time15;

AssemblyWorkstation2.Capactiy = 1;

AssemblyWorkstation2.Assemble = true;

Since queues are all in the FIFO rule, capacity is *capacity1*, the model is rather simple:

New PE: Queue

Queue.Capacity = capcity1;

The system-level modeling is done by applying Node-Based strategy, shown here:

GENERATOR: g1,g2,g3,g4,g5,g6,g7,g8,g9; // suppose they are predefined

Q1.from (g1);

Q2.from (g2);

Q3.from (g3);

Q4.from (g4);

Q6.from (g5);

Q7.from (g6);

Q8.from (g7);

```
Q9.from (g8);

Q12.from (g9);

WeldingStation1.from (Q1,Q2,Q3);

WeldingStation2.from (Q4,Q5,Q6);

WeldingStation3.from (Q7,Q8);

InspectionStation1.from (Q9);

InspectionStation2.from (Q12);

AssemblyStation1.from (Q10,Q11,Q13);

AssemblyStation2.from (Q14, Q15);

Q5.from (WeldingStation1);

Q13.from (WeldingStation3);

Q11.from (InspectionStation2);

Q10.from (InspectionStation1);

Q15.from (WeldingStation2);

Q14.from (AssemblyStation1);

END
```

When the production line illustrated here is studied as a PE of the whole production line, it can be contained in a loosely coupled PE. However, the generators are not processors, therefore, they cannot be included. The CPE_t is shown below:

```
New CPE(tc = false): AssemblyLine;
    //first, encapsulate the production line
    AssemblyLine.Output_Port.from (AssemblyStation2);
    Q1.from (AssemblyLine.Input_Port[1]);
    Q2.from (AssemblyLine.Input_Port[2]);
    Q3.from (AssemblyLine.Input_Port[3]);
    Q4.from (AssemblyLine.Input_Port[4]);
    Q6.from (AssemblyLine.Input_Port[5]);
    Q7.from (AssemblyLine.Input_Port[6]);
    Q8.from (AssemblyLine.Input_Port[7]);
    Q9.from (AssemblyLine.Input_Port[8]);
    Q12.from (AssemblyLine.Input_Port[9]);
    //encapsulation finished
    //now, connect this CPE to others.
    AssemblyLine.Input_Port[1].from(g1);
    AssemblyLine.Input_Port[2].from(g2);
    AssemblyLine.Input_Port[3].from(g3);
    AssemblyLine.Input_Port[4].from(g4);
    AssemblyLine.Input_Port[5].from(g5);
    AssemblyLine.Input_Port[6].from(g6);
    AssemblyLine.Input_Port[7].from(g7);
```

```
AssemblyLine.Input_Port[8].from(g8);

AssemblyLine.Input_Port[9].from(g9);
```

Let us suppose the next resource connecting to this assembly line is another assembly line called PackageLine, the system level modeling is:

```
PackageLine.Input_Port[1].from(AssemblyLine.Output_Port);
```

The hierarchical modeling provides the users flexibilities to view and model the real world systems in arbitrary level of detail. The Node-Based strategy provides a flexible routing procedure.

CHAPTER FIVE

DISCUSSION, FUTURE WORK AND CONCLUSIONS

5.1 Discussion

In this thesis, the author is working towards a conceptual framework for a distributed Object-Based industrial simulation system to lay the groundwork for the Intelligent Simulation System.

In the abstract, the author argues that DOBIS can outperform existing simulation systems in several aspects. Indeed, the ways that DOBIS can outperform other systems include: functional performance (the things it can do), usability (ease with which it can be instructed to do things), consistency with mental models (it lets people model systems however they picture the system rather than teaching them to think in a certain way), and flexibility & extensibility. Another point here regards the ability of most other packages to let users write extra routines that plug into the package for specialized computation. Providing this facility is one thing. But creating the kind of open architecture that the author is doing with DOBIS elevates this ability far beyond anything that just 'plug-ins' can provide.

In order to achieve high levels of flexibility, it is essential to simplify systems that "appear" to be complicated. However, no system is simple when high levels of detail are considered. This is the reason why existing general-purpose simulation systems have

hundreds of commands in order to fit in as many situations as possible.

Imagine a factory floor where the flows of entities and the directions of flows represent the dependencies between machines. Machines are working simultaneously, which is said to be Parallel Activities. Note that this is at the system level, which means a machine is an object. When a workpiece arrives at a certain machine, it generates a set of flows inside the machine that activate all the internal operations. Now, the machine becomes the system and the internal operation becomes the object. Depending on the required level of detail, the internal operations could become systems. This kind of transition indicates that the internal operation, object and system are different views of the same thing. Therefore they can be modeled by a basic element that consists of both system-level and object-level information, and such basic elements can be hierarchically structured to model complex resources with flexible resolution. In this way, the structure of a complex system can be clearly identified. It also makes it possible to develop a generic element that is able to capture the properties of all the resources in the system, hence reduce the number of commands.

In this light, the author first developed the Node-Based modeling strategy. This strategy combines the Event-Scheduling strategy and the Object-Oriented paradigm, focuses on the interactions between resources. A node is a generic form that is used to model basic forms of resources, which can be either a system, a single machine, or even a function of a resource. At system-level modeling, every resource in the system is represented by a node. The system is hence translated into a network of nodes. Therefore routes of flow entities are clearly identified. It eliminates the dependency on the flow

entities that limit the ability to model parallel activities. It is also closer to reality where resources control the flow. Most importantly, it leads to a uniform object structure and provides the platform for the development of the Proto-Element.

The Node-Based strategy provides primitive modeling abilities and a platform for the Proto-Element.

Node-Based strategy focuses on the interactions between resources and provides a generic form for all kinds of resources. Based on the form of Node, Proto-Element (PE) inherits important features from Petri-Nets to model properties of resources, specifically the performance of a system, which is *time* and *capacity*. To model complex systems with parallel activities, the logical relationships between resources are captured using generic logical operators. This extension brings out the Coupled Proto-Element (CPE). There are two kinds of CPEs, one is called tightly coupled, which is coupled by logical relationships. It is also referred to as explicit logical relationship. The other one is called loosely coupled, where the relationship between two PEs are maintained by the flow of entities. It is also referred to as implicit logical relationship by the author. A typical application of loosely Coupled Proto-Element is to encapsulate a subsystem to a single element.

To provide formal representation, the author mapped PE and CPE into the well-known and proven Parallel DEVS formalism. It also proves that the PE/CPE approach utilizes the hierarchical modeling methodology. This PE/CPE approach significantly improves the modeling ability and lowers the complexity of syntax, which results in shortened learning period and lowered maintenance cost.

The uniform object structure, and the ability of capturing the relationships between objects make the PE/CPE approach high levels of flexibility to model complex systems.

Hierarchical Modeling approach has shown high levels of modeling ability and flexibility in DEVS. The Parallel DEVS is a well-known hierarchical modeling formalism; it is a proven approach studied by many researchers and simulation practitioners [2]. By mapping the CPE into the Parallel DEVS, the new framework is hence supported by a formal hierarchical modeling approach. The benefits of applying this approach are illustrated in Chapter 3 and 4.

There are six major benefits in this approach: first, the simulation system is able to build dynamical models with changing scales; secondly, it is easy to model and manage large-scale simulation models; thirdly, it is able to capture any level of detail; fourth, it eases the work of validation and verification; fifth, it is able to implement both top-down and bottom-up modeling; sixth, which is unique to the author's work, models at different levels may have the same structures, therefore it is possible to reuse the model structures with different parameters.

The Proto-Element applied in this approach is not a full-fledged construction yet. Because the structure that has been discussed up to now are from the preliminary coding done by the author that is used to prove the concept. However, from the author's point of view, extending the current PE structure to a fully developed structure will not be a hard task. Because the basic concept of the PE is to capture the generic attribute of objects. Under the DEVS formalism, it is to find out the possible generic set of states a certain set of

objects in an area of study can possess.

In this thesis, the author considers the logical relationships between objects are static in the system. However, in real world, other than static relationships, there are dynamic relationships, e.g., a relationship based on a if/then rule, e.g., a failure of one machine signals another machine to start working (redundant configuration). The author recommends to construct CPE according to the static relationships. This is because the PEs represent processes, and these processes are seen as the flows of the entities. Therefore, the connections of the PEs also represent the physical connections of the process within an object. Dynamic relationships should be addressed in the future work.

From the beginning, Object-Oriented paradigm was applied as an essential design requirement of the framework. This not only makes the modeling abstraction task easier, but also eases the programming task. Object-Oriented abstraction strategy provides an intuitive way to model real world objects. It also provides a user-friendly environment for modeling and programming. Users are able to choose the working environment they are familiar with, e.g., Visual Cafe, Borland J++, etc. Java programming language is a native Object-Oriented programming language. By employing Java, platform independent ability is achieved, which will make distributed computing over heterogeneous network possible.

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artefacts of software systems, as well as for business modeling and other non-software systems [88]. It is a generic modeling system that the author only became aware of late in the thesis and were unable to consider as fully as he

should have. However, UML has a set of different application focuses (software engineering) than the DOBIS does. The author acknowledges that it should be studied as part of future work, and it could contribute significantly to specifying the DOBIS framework to facilitate implementation.

5.2 Future Work

The author discussed and proposed a framework for modeling. The author is not developing the whole system or a "perfect" system in this thesis. Instead, the conceptual framework is just the first step towards a working system. What the author has done in this thesis is prove of theory. Actual implementation, coding and testing have to be conducted by other research assistants directed by Dr. Salustri. Since this thesis concentrates on modeling, there are several issues not within the scope of the author's work, which have not been answered and which have to be explored in future work.

One issue mentioned by the author was the design of a GUI (Graphical User Interface) for the modeling environment. In this thesis, the author proposed the possible form of the modeling language, which is text-based. However, for a complex model, where there are hundreds of lines of modeling information, it is better to have a GUI so that all the information is visually available to the user. The benefit of a GUI system has been proved by many systems. In the design of the form of the modeling language, the author exercised Object-Oriented paradigm, which makes the implementation of GUI a straightforward task. For example, the line "AssemblyLine.Input_Port[8].from(g8);" means the PE named

AssemblyLine has a connection from PE g8 to its eighth input port. In the GUI, it is possible to draw two boxes, give them the names above, connect them together by an arrowed line through drag and drop, and put a text beside the arrow indicating the "Input_Port[8]". See Figure 5.1,

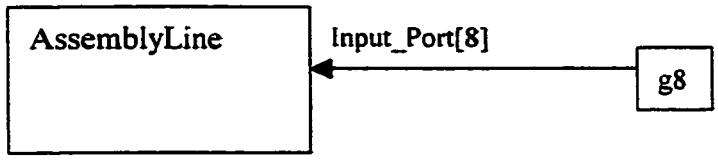


Figure 5.1 GUI example

Users should also have the option to build models that are bi-directional, which means either input text to generate graphics or generate graphics and output text model information. Mappings between visual data and text-based modeling language are easy through Object-Oriented paradigm. The GUI and text-based interface should work as layers on top of the model. See Figure 5.2,

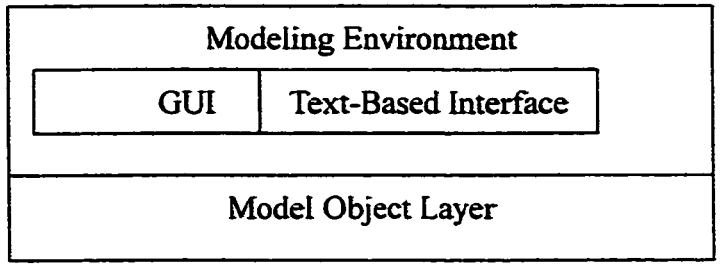


Figure 5.2 Modeling environment and model object layers

This modeling environment structure is being used by many existing systems, e.g., SIMAN, AutoMod, etc.

DOBIS uses time and capacity to model the performances of resources, and only linear mathematical problems have been considered. This is not enough in some cases. For example, the Automatic Guided Vehicle (AGV). For high-resolution model, the non-linear properties of AGV have to be modeled, e.g., the accelerations during the start and stop. A very good modeling approach for this kind of resource can be found in AutoMod. However, this would make the object structure specific, which contradicts the author's original intention of making the PE generic and thus downgrade the system's flexibility. To solve this problem, certain kind of mathematical approach has to be applied, and this approach should be implemented by the polymorphism property of Object-Oriented programming language. From the author's point of view, adding an attribute to the PE's structure could do this.

To model computer controlled automation lines, there has to be a control module built into the system. Currently, DOBIS is able to accomplish simple control commands, e.g., distinguishing part types and apply different processes according to part properties. This kind of control can be characterized as stand-alone, where one machine cannot access information outside itself. Therefore the control module should work on top of the PEs, and there should be an interface in the PE that bi-directionally interacts with the module, so that the control module is able to control resources according to the overall state of the system. See Figure 5.3,

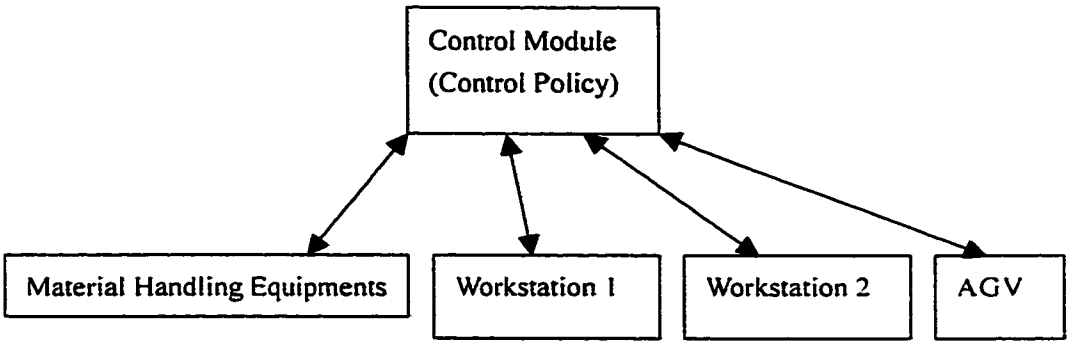


Figure 5.3 Control module and resources

An immediate application of this control module is for the route planning of AGVs. In this application, the AGV has to gather information on the workstations and then plan an optimal route. Another application is in the FMS where central controllers generate production plans according to run-time information gathered from all the workstations.

Another issue is the distributed computing. In this thesis, the author distributed computing aspect of the framework is rather conceptual. That is, by 'conceptual' the author means that he is defining an implementation that reflects the needs of industrial engineers at the conceptual level and then implementing them using whatever technologies are available. Further, the end users, who this framework wants to support, are not interested in, say, the details of how the object system distributes over multiple machines or over the Internet. The goal is to define something most likely to be usable by engineers, instead of something of particular elegance to computer scientists. Also, the author looks at the DOBIS as a 'requirements specification' for computer scientists working in distributed simulation. It's up to them to work out the implementation details.

The clock synchronizing algorithm and the distributed computing framework used in the illustration in this thesis are certainly not the best for the application discussed. However, the developments in computer science as well as the mathematical algorithms have shown that there are always better algorithms. Therefore, in the future work, the importance of developing an adaptable framework is more than that of looking for a "best" algorithm.

5.3 Conclusions

In the beginning of the thesis, the author reviewed current researches in simulation, existing simulation systems, simulation modeling strategies, methodologies and Object-Oriented paradigm. The author then identified goals to achieve for the new framework: flexible, easy to use. In order to achieve these goals, detailed implementation requirements were drawn out: utilizing hierarchical modeling, Object-Oriented abstraction and Event-Scheduling strategy. These brought out the core of DOBIS, which is PE and CPE: a modeling strategy developed by the author that extends Parallel DEVS, inherits the performance modeling strategy from Petri Nets and clarifies logical relationships between parallel processes. A two-layer construction framework is also discussed, this framework pushes down much of the tedious "low-level" aspects of simulation to the systems level, and away from the user.

In this section, the author summarizes the achievement in this thesis.

Node-Based Modeling Strategy

This strategy applies a basic element that is able to model both system-level and object-level information. Such basic elements can be hierarchically structured to model complex resources with flexible resolution.

The Node-Based strategy provides primitive modeling abilities and a platform for the Proto-Element.

Proto-Element and Coupled Proto-Element

As core of the DOBIS, they are based on the form of Node. They model properties of resources by *time* and *capacity*. To model complex systems with parallel activities, the logical relationships between resources are captured using generic logical operators. This extension brings out the Coupled Proto-Element (CPE). There are two kinds of CPEs: tightly coupled and loosely coupled.

To provide formal representation, the author mapped PE and CPE into the well-known and proven Parallel DEVS formalism.

The uniform object structure, and the ability of capturing the relationships between objects make the PE/CPE approach high levels of flexibility to model complex systems.

Hierarchical Modeling

Hierarchical Modeling approach has shown high levels of modeling ability and flexibility in DEVS. By mapping the CPE into the Parallel DEVS, the new framework is hence supported by a formal hierarchical modeling approach.

Object-Oriented Modeling, Platform Independent and Friendly Working Environment

As an essential part of the DOBIS, Object-Oriented paradigm is applied in simulation object abstraction process and programming. This not only makes the modeling abstraction task easier, but also eases the programming task.

APPENDIX A

GPSS model for the problem discussed on page 96.

```
Transit TABLE M1,200,200,20

GENERATE (Exponential(1,0,300)) ;New order arrives

SPLIT 2,Factory,1 ;Make 2 copies of order

QUEUE Motor ;Queue for motor

SEIZE Motor ;Get a Facility

DEPART Motor ;Depart the queue

ADVANCE 200,100 ;Take motor from stock

RELEASE Motor ;Free the Facility

TRANSFER ,Tryout ;Send to trial assembly

Factory TEST E P1,2,Baseplate ;Is P1=2 ?

QUEUE Pumps ;Join the Queue (P1=2)

SEIZE Pumps ;Get a Facility

DEPART Pumps ;Depart the Queue

ADVANCE 180,120 ;Prepare the Pump

Pump MATCH Plate ;Wait for baseplate

ADVANCE 50,10 ;Check pump on baseplate

RELEASE Pumps ;Free the Facility
```

```
TRANSFER ,Tryout ;Send for a tryout

Baseplate QUEUE Base ;Join Queue P1 must=3

SEIZE Base ;Get a Facility

DEPART Base ;Depart the Queue

ADVANCE 80,20 ;Make the baseplate

Plate MATCH Pump ;Wait for the pump unit

ADVANCE 50,10 ;Check the pump on

; baseplate

RELEASE Base ;Free the Facility

Tryout GATHER 3 ;Gather 3 units to tryout

ADVANCE 60 ;Trial assembly

TEST E P1,1,Finish ;Is it the motor?(P1=1)

SEIZE Paint1 ;Get first paint Facility

ADVANCE 100,20 ;Paint the motor

RELEASE Paint1 ;Free paint Facility 1

TRANSFER ,Build ;Send for assembly

Finish TEST E P1,2,Basplate ;Is it the pump?(P1=2)

SEIZE Paint2 ;Get paint Facility 2

ADVANCE 120,30 ;Paint the Pump

RELEASE Paint2 ;Free paint Facility 2

TRANSFER ,Build ;Send for assembly
```

```
Basplate SEIZE Galvanize ;Get a Facility

ADVANCE 120,30 ;Galvanize baseplate

RELEASE Galvanize ;Free the Facility

Build ASSEMBLE 3 ;Collect 3 units

ADVANCE 150,30 ;Assemble unit

TABULATE Transit ;Record transit time

TERMINATE 1 ;One unit completed
```

REFERENCES

[1] Gerald W. Evans, William E. Biles, *Simulation of Advanced Manufacturing Systems*. Proceedings of the 1992 Winter Simulation Conference

[2] Narayanan. S., D. A. Bodner, U. Sreekanth, T. Govindaraj, L. F. McGinnis, and C. M. Mitchell, *Research in Object-Oriented Manufacturing Simulations: An Assessment of the State of the Art*, Technical Report MHRC TR-94-03, Material Handling Research Centre, Georgia Institute of Technology, 1994

[3] Joseph G. Macro, Duane L. Setterdahl, *Establishing An Object-Oriented Methodology For The Simulation and Control of Intergrated Manufacturing Systems*. Proceedings of the 1994 Winter Simulation Conference

[4] John P. Shewchuk, Tien-Chien Chang, *An Approach to Object-Oriented Discrete-Event Simulation of Manufacturing Systems*, Proceedings of the 1991 Winter Simulation Conference

[5] John B. Evans, *Structures of Discrete Event Simulation, an introduction to the engagement strategy*, Ellis Horwood series in artificial intelligence, 1988

[6] Bernard P. Zeigler, *Theory of Modelling and Simulation*, John Wiley&Sons Inc., 1976

[7] David J. Thuente, *Critique of SIMAN As A Programming Language*, proceedings of 15th annual conference on Computer Science, Page 385

[8] Paul G. McConnell, D. J. Medeiros, *Real-Time Simulation For Decision Support In Continuous Flow Manufacturing Systems*, Proceedings of the 1992 Winter Simulation

Conference

[9] Giorgio Bruno and Alessandro Balsamo, *Petri Net-Based Object-Oriented Modelling of Distributed Systems*, Conference proceedings on Object-oriented programming systems, languages and applications, 1986, Pages 284-293

[10] Hemant C. Bhuskute, Manoj N. Duse, Jagannath T. Gharpure, David B. Pratt. Manjunath Kamath, Joe H. Mize, *Design And Implementation of A Highly Reusable Modeling and Simulation Framework For Discrete Part Manufacturing Systems*. Proceedings of the 1992 Winter Simulation Conference

[11] Jeffrey A. Joines, Kenneth A. Powell, Jr., Stephen D. Roberts, *Object-Oriented Modeling and Simulation with C++*, Proceedings of the 1992 Winter Simulation Conference

[12] Joel J. Luna, *Hierarchical, Modular Concepts Applied to An Object-Oriented Simulation Model Development Environment*, Proceedings of the 1992 Winter Simulation Conference

[13] Diane P. Bischak, Stephen D. Roberts, *Object-Oriented Simulation*, Proceedings of the 1991 Winter Simulation Conference

[14] Henricus J.E. Verhagen, *Agents and Sociality*, licenciate thesis, June 1998

[15] Georgios Theodoropoulos and Brian Logan, *A Framework for the Distributed Simulation of Agent-Based Systems*, 13th European Simulation Multiconference (ESM99), Warsaw, Poland, June 1-4, 1999

[16] S. Flake, Ch. Geiger, G. Lehrenfeld, W. Mueller, V. Paelke. *Agent-Based Modeling for*

Holonic Manufacturing Systems with Fuzzy Control, NAFIPS'99, 18th International Conference of the North American Fuzzy Information Processing Society, New York, USA, June 10-12, 1999

[17] Donald O. Hamnes, Anand Tripathi, *Investigations in Adaptive Distributed Simulation*, Proceedings of the 1994 workshop on Parallel and distributed simulation, 1994, Pages 20 - 23

[18] Andre M.C. Campos, David R.C. Hill, *Web-Based Simulation Of Agent Behaviors*, In Proceedings of the 1998 WEBSIM Conference, San Diego, California

[19] J. Robert Ensor and Gianpaolo U. Carraro, Peloton, *A Distributed Simulation For The World Wide Web*, 1998 International Conference On WebBased Modeling and Simulation. Simulation Series, Vol. 30, No. 1. Society for Computer Simulation International, San Diego (1998) 159-164

[20] Ernest H. Page, 1998, *The Rise of Web-Based Simulation: Implications for the High Level Architecture*, In Proceedings of the 1998 Winter Simulation Conference, pages 1663--1668, Washington, DC

[21] G. Chiola, A. Ferscha, *A Distributed Discrete Event Simulation Framework for Timed Petri Net Models*, Technical Report Series of the Austrian Centre for Parallel Computation, ACPC/TR 93-21, December 1993

[22] Alois Ferscha, Satish K. Tripathi, *Parallel and Distributed Simulation of Discrete Event Systems*, Parallel and Distributed Computing Handbook, pages 1003 -- 1041. McGraw-Hill, 1996

[23]Susumu Fujii, Haruhisa Tsunoda, Atsusi Ogita, Yasusi Kidani, *Distributed Simulation Model For Computer Integrated Manufacturing*, Proceedings of the 1994 Winter Simulation Conference

[24]G. S. Thomas, *Parallel Simulation of Petri Nets*, Tech. Rep. TR 91-05-05, Dept. of Computer Science, Univ. of Washington. May 1991

[25]Jeff S. Steinman, *Breathing Time Warp*, Proceedings of the 1993 workstop on Parallel and distributed simulation, May 16-19, 1993, San Diego, CA USA

[26]David Jefferson, *Distributed Simulation and the Time Warp Operating System*, Proceedings of the Eleventh ACM Symposium on Operating systems principles. November 8 –11, 1987, Austin, TX USA

[27]Yi-Bing Lin and Edward D. Lazowska, *A Study of Time Warp Rollback Mechanisms*, ACM Transactions on Modeling and Computer Simulation, Volume 1, Issue 1, 1991

[28]Pete Ball, Doug Love, *The Key to Object-Oriented Simulation: Separating the User and the Developer*, Proceedings of the 1995 Winter Simulation Conference

[29]C. Dennis Pegden, Deborah A. Davis, *Arena: A SIMAN/Cinema-Based Hierarchical Modeling System*, Proceedings of the 1992 Winter Simulation Conference

[30]Chenho Kung, *The OO paradigm*, Encyclopedia of Microcomputers, Vol. 12, pp. 287 - 305, Marcel Dekker Publishing Inc., 1993

[31]Rechard Scott Brink, *A Petri Net Design, Simulation, and Verification Tool*, A Master's Thesis, September, 1996

[32]Zeigler, B. P, *Multifaceted Modelling and Discrete Event Simulation*, Academic Press,

London, 1984

[33] Hyup J. Cho and Young K. Cho, *DEVS-C++ Reference Guide*, 31 October 1997

[34] Francis Neelamkaril, *Computer Simulation and Modeling*, John Wiley&Sons, 1987

[35] John A. Hamilton, Jr., David A. Nash, Udo W. Pooch, *Distributed Simulation*, CRC Press Computer Engineering Series, March, 1997

[36] Sun Microsystems Inc., *Java 2 SDK, Standard Edition Documentation, Version 1.3*, http://java.sun.com/docs, 2000

[37] AutoSimulations, Inc., *AutoMod User's Manual*, June 1999

[38] Julio C. Martinez, Photios G. Ioannou, *Advantages of the Activity Scanning Approach in the Modeling of Complex Construction Processes*, Proceedings of the 1995 Winter Simulation Conference

[39] Tillal Eldabi, Ray J. Paul, *Flexible Modeling of Manufacturing Systems with Variable Levels of Detail*, Proceedings of the 1997 Winter Simulation Conference

[40] Grace Y. Lin, James J. Solberg, *An Agent-Based flexible Routing Manufacturing Control Simulation*, Proceedings of the 1994 Winter Snnulation Conference, 1994

[41] Merriam-Webster Inc., *Merriam-Webster Online*, http://www.m-w.com/, 2001

[42] Arun S. Kashyap, Suresh K. Khator, *Modeling of A Tool Shared Flexible Manufacturing System*, Proceedings of the 1994 Winter Simulation Conference

[43] ZDnet and internet.com Corporation, *ZDWEBOPEDIA*, http://www.zdwebopedia.com/, 2000

[44] Frank Chance, Jennifer Robinson, John Fowler, *Supporting Manufacturing With*

Simulation: Model Design, Development, and Deployment, Proceedings of the 1996 Winter Simulation Conference

[45] Stewart Robinson, *Simulation Model Verification and Validation: Increasing the Users' Confidence*, Proceedings of the 1997 Winter Simulation Conference

[46] Nirupama Nayani, Mansooreh Mollaghasemi, *Validation and Verification of the Simulation Model of a Photolthography Process in Semiconductor Manufacturing*, Proceedings of the 1998 Winter Simulation Conference

[47] Robert G. Sargent, *Validation and Verification of Simulation Model*, Proceedings of the 1999 Winter Simulation Conference

[48] Osman Balci, *Validation, Verification, and Testing Techniques Throughout the Life Cycle of A Simulation Study*, Proceedings of the 1994 Winter Simulation Conference

[49] Jerry Banks, *Getting Started With AutoMod, AutoSimulations Inc.*, August 2000

[50] Paul K. Davis, Richard Hillestad, *Families of Models That Cross Levels of Resolution: Issues For Design, Calibration and Management*, Proceedings of the 1993 Winter Simulation Conference

[51] Fernando J. Barros, Bernard P. Zeigler, Paul A. Fishwick, *MultiModels and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM*, Proceedings of the 1998 Winter Simulation Conference.

[52] Martha A. Centeno,Charles Standridge, *Modeling Manufacturing Systems: An Information-Based Approach*, Proceedings of the 24th Annual Symposium 1991. SCS Eastern Multi Conference, A.H. Ruttan (ed), 230-239.

[53] Allan Carrie, *Simulation of Manufacturing Systems*, John Wiley & Sons, 1988

[54] David B. Pratt, Phillip A. Farrington, Chuda B. Basnet, Hemant C. Bhuskute, Manjunath Kamath, Joe H. Mize, *A Framework for Highly Reusable Simulation Modeling: Separating Physical, Information, and Control Elements*, The 24th Annual Simulation Symposium, pages 254--261, New Orleans, Louisiana, 1991. IEEE Computer Society Press

[55] Minuteman Software, *GPSS World Reference Manual*, web address: http://www.minutemansoftware.com/reference/reference_manual.htm, 2000

[56] Jeffrey A. Joines, Stephen D. Roberts, *Design of Object-Oriented Simulations in C++*, Proceedings of the 1996 Winter Simulation Conference.

[57] Paul A. Fishwick, Richard E. Nance, Jeff Rothenberg, Robert G. Sargent, *Discrete Event Simulation Modeling: Directions for the '90s*, Proceedings of the 1992 Winter Simulation Conference

[58] Thorsten Daum, Robert G. Sargent, *A Java Based System for Specifying Hierarchical Control Flow Graph Models*, Proceedings of the 1997 Winter Simulation Conference

[59] Douglas G. Fritz, Robert G. Sargent, *An Overview of Hierarchical Control Flow Graph Model*, Proceedings of the 1995 Winter Simulation Conference

[60] Douglas G. Fritz, Robert G. Sargent, Thorsten Daum, *An Overview of HI-MASS (Hierarchical Modeling and Simulation System)*, Proceedings of the 1995 Winter Simulation Conference

[61] Joe H. Mize, David H. Withers, Bernard P. Zeigler, *Hierarchical Modeling for*

Discrete Event Simulation, Proceedings of the 1993 Winter Simulation Conference

[62] Thomas M. O'Donovan, *GPSS, Simulation made Simple*, John Wiley & Sons Ltd, 1979.

[63] Richard Kilgore and Kevin Healy, *Java, Enterprise Simulation and the SILK Simulation Language*, Proceedings of the 1998 SCS International Conference on Web-Based Modeling and Simulation

[64] John S. Carson, *Modeling and Simulation Worldviews*, Proceedings of the 1993 Winter Simulation Conference

[65] Raid Al-Aomar, Daniel Cook, *Modeling At The Machine-Control Level Using Discrete Event Simulation*, Proceedings of the 1998 Winter Simulation Conference

[66] Herb Schwetman, *Object-Oriented Simulation Modeling With C++/CSIM17*, Proceedings of the 1995 Winter Simulation Conference

[67] Thorsten Daum, Robert G. Sargent, *Scaling, Hierarchical Modeling, and Reuse In An Object-Oriented Modeling and Simulation System*, Proceedings of the 1999 Winter Simulation Conference

[68] Paul A. Fishwick, *A Simulation Environment for Multimodeling*, Discrete Event Dynamic Systems: Theory and Applications, 3:151--171, 1993.

[69] Miryam Barad, *Timed Petri Nets As A Verification Tool*, Proceedings of the 1998 Winter Simulation Conference

[70] C. Chaouiya, Y. Dallery, *Petri Net Models of Pull Control Systems for Assembly Manufacturing Systems*, Second Int. Workshop on Manufacturing and Petri Nets,

ICATPN,Toulouse 97

[71] Bernd Däne, Angela Mölders, Andrea Melber, Wolfgang Fengler, *Modeling an Industrial Transportation Facility with Coloured Petri Nets*, Workshop for Manufacturing and Petri Nets, Toulouse, June 23-27, 1997

[72] R. Valette, H. Pingaud, A. Pages, D. Andreu, J.C. Pascal, *Modeling, Simulation And Control Of Event-Driven Operations In Process Systems*, INRIA-IEEE Symposium on Emerging Technologies and Factory Automation, ETFA'95, vol. 3, pp. 119-128, Paris, October 1995

[73] Lakos CA, *The Object Orientation of Object Petri Nets*, Workshop on Object Oriented Programming and Models of Concurrency, Torino, Italy, 1995

[74] Armin Zimmermann, *Modeling of Manufacturing Systems and Production Routes Using Colored Petri Nets*, Proc. of the 3rd IASTED Int. Conf. on Robotics and Manufacturing, Cancún, Mexico, pp. 380–383, 1995

[75] K. Thirunarayan, G. Kniesel, and H. Hampapuram, *Simulating Multiple Inheritance and Generics in Java*, to appear in: Computer Languages, 2001.

[76] Luca Cardelli, *A Semantics of Multiple Inheritance*, Information and Computation 76, 138-164, 1988

[77] Yen-Ping Shan, Tom Cargill, Brad Cox, William Cook, Mary Loomis, Alan Snyder, *Is Multiple Inheritance Essential to OOP?* Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications September 26 - October 1, 1993, Pages 360 – 363,Washington United States

[78] Kevin J. Healy, Richard A. Kilgore, *Introduction To Silk And Java-Based Simulation*, Proceedings of the 1998 Winter Simulation Conference, 1998

[79] Jeffrey A. Joines, Stephen D. Roberts, *An Introduction To Object-Oriented Simulation In C++*, Proceedings of the 1997 Winter Simulation Conference

[80] Tag Gon Kim and Bernard P. Zeigler, *The DEVS Formalism: Hierarchical, Modular Systems Specification In An Object Oriented Framework*, Proceedings of the 1987 conference on Winter simulation

[81] Alex ChungHen Chow, Bernard P, Zeigler, *Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism*, Proceedings of the 1994 Winter Simulation Conference

[82] Fillipo A. Salustri, *Intelligent Simulation*, web address: http://deed.ryerson.ca/~fil/projects/#isim, 2000

[83] Minuteman Software, *GPSS World Tutorial Manual*, Minuteman Software, 2000

[84] Minuteman Software, *A Comparison of SLX and GPSS/H*, Minuteman Software, 2000

[85] Richard M. Fujimoto, *Parallel Discrete Event Simulation*, Communications of the ACM, Volume 33, Issue 10, 1990

[86] Om P. Damani, Yi-Min Wang, Vijay K. Garg, *Optimistic Distributed Simulation Based on Transitive Dependency Tracking*, Proceedings of the 1997 workshop on Parallel and distributed simulation, June 10 - 13, 1997, Austria

[87] Jerry Banks, *Handbook of Simulation*, John Wiley & Sons, Inc., 1998

[88] Rational Software, *Unified Modeling Language*, www.rational.com

VITA AUCTORIS

Bo Gao was born in 1973 in Beijing, P.R.China. He graduated from the No.55 high school in 1992. From there he went to the Beijing Polytechnic University where he obtained a B.Eng. in Mechanical Engineering in 1997. He is currently a candidate for the Master's degree in Industrial Engineering at the University of Windsor and hopes to graduate in Fall 2001.